EDUCACIÓN

SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Orizaba

"2019, Año del Caudillo del Sur, Emiliano Zapata"

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

OPCIÓN I.- TESIS

TRABAJO PROFESIONAL

"OPTIMIZACIÓN DE LA DERIVACIÓN DE PRODUCTOS EN MULTILÍNEAS DE PRODUCTOS DE SOFTWARE UTILIZANDO INGENIERÍA DE SOFTWARE BASADA EN BÚSQUEDA".

QUE PARA OBTENER EL GRADO DE:

DOCTORA EN CIENCIAS DE LA INGENIERÍA

PRESENTA:

M.S.C. Guadalupe Isaura Trujillo Tzanahua

DIRECTOR DE TESIS:

Dr. Ulises Juárez Martínez

CODIRECTOR DE TESIS:

Dr. Alberto Alfonso Aguilar Lasserre

INSTITUTO TECNOLÓGICO DE ORIZABA

ORIZABA, VERACRUZ, MÉXICO. **NOVIEMBRE 2019**

EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Orizaba

"2019, Año del Caudillo del Sur, Emiliano Zapata"

FECHA: 12/11/2019
DEPENDENCIA: POSGRADO
ASUNTO: Autorización de Impresión
OPCIÓN: I

C. **GUADALUPE ISAURA TRUJILLO TZANAHUA**
CANDIDATO A GRADO DE DOCTOR EN:
CIENCIAS DE LA INGENIERIA

De acuerdo con el Reglamento de Titulación vigente de los Centros de Enseñanza Técnica Superior, dependiente de la Dirección General de Institutos Tecnológicos de la Secretaría de Educación Pública y habiendo cumplido con todas las indicaciones que la Comisión Revisora le hizo respecto a su Trabajo Profesional titulado:

"OPTIMIZACION DE LA DERIVACION DE PRODUCTOS EN MULTILINEAS DE PRODUCTOS DE SOFTWARE UTILIZANDO INGENIERIA DE SOFTWARE BASADA EN BUSQUEDA".

Comunico a Usted que este Departamento concede su autorización para que proceda a la impresión del mismo.

A T E N T A M E N T E

MARIO LEONCIO ARRIOJA RODRIGUEZ
JEFE DE LA DIV. DE ESTUDIOS DE POSGRADO

EDUCACIÓN | TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE ORIZABA
DIVISIÓN DE ESTUDIOS PROFESIONALES

Avenida Oriente 9 Núm. 852, Colonia Emiliano Zapata, C.P. 94320 Orizaba, Veracruz, México
Tel. 01 (272) 7 24 40 96, Fax. 01 (272) 7 25 17 28 e-mail: orizaba@itorizaba.edu.mx
www.orizaba.tecnm.mx

EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Orizaba

"2019, Año del Caudillo del Sur, Emiliano Zapata"

FECHA : 17/10/2019

ASUNTO: **Revisión de Trabajo Escrito**

C. MARIO LEONCIO ARRIOJA RODRIGUEZ
JEFE DE LA DIVISION DE ESTUDIOS
DE POSGRADO E INVESTIGACION.
P R E S E N T E

Los que suscriben, miembros del jurado, han realizado la revisión de la Tesis del (la) C. :

GUADALUPE ISAURA TRUJILLO TZANAHUA

la cual lleva el título de:

"OPTIMIZACION DE LA DERIVACION DE PRODUCTOS EN MULTILINEAS DE PRODUCTOS DE SOFTWARE UTILIZANDO INGENIERIA DE SOFTWARE BASADA EN BUSQUEDA".

Y concluyen que se acepta.

A T E N T A M E N T E

PRESIDENTE	: DR.	**ULISES JUAREZ MARTINEZ**	FIRMA
SECRETARIO	: DR.	**ALBERTO ALFONSO AGUILAR LASSERRE**	FIRMA
PRIMER VOCAL	: DR.	**GINER ALOR HERNANDEZ**	FIRMA
SEGUNDO VOCAL	: DR.	**GUILLERMO CORTES ROBLES**	FIRMA
TERCER VOCAL	: DR.	**CUAUHTEMOC SANCHEZ RAMIREZ**	FIRMA
VOCAL SUP.	: DRA.	**MARIA KAREN CORTES VERDIN**	FIRMA

EGRESADO(A) DEL DOCTORADO EN **CIENCIAS DE LA INGENIERIA**

OPCION: **I** **Tesis**

Agradecimientos

A Dios, por guiarme en todo momento y por permitirme alcanzar una nueva meta.

Al Dr. Ulises Juárez Martínez por aceptarme para realizar esta tesis doctoral bajo su dirección. Su apoyo, confianza en mí y guía han sido un aporte invaluable para el desarrollo de esta investigación y para mi formación profesional.

Al Dr. Alberto Alfonso Aguilar Lasserre por brindarme su amistad y apoyo durante todo este tiempo. Su asesoría y excelentes ideas contribuyeron a ..este proyecto.

De manera especial, a la Dra. Karen Cortés Verdín por su colaboración, retroalimentación y valiosas aportaciones que contribuyeron al enriquecimiento de esta investigación.

A los miembros del jurado, Dr. Giner Alor Hernández, Dr. Guillermo Cortés Robles y Cuauhtémoc Sánchez Ramírez por su tiempo y apoyo para revisar la tesis y que con sus valiosos comentarios me ayudaron a mejorar este trabajo de tesis.

A todos los maestros que han contribuido a lo largo de mi formación académica.

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo de becas para la realización de mis estudios de posgrado.

A la persona más importante de mi vida, mi mamá Rufina. Por todo el amor, cuidados, sabios consejos en esos momentos difíciles y apoyo que me ha dado en las diferentes etapas de mi vida.

A mi hermano Miguel, que con su amor y comprensión me ha alentado a seguir adelante.

A mi tía Mari, porque en los momentos más difíciles ha estado conmigo y que ha sido como una segunda madre.

A todas las personas que directa e indirectamente me brindaron su apoyo, colaboración y amistad.

Índice general

Índice de figuras

Índice de tablas

Resumen

La Ingeniería de Software incorpora prácticas o metodologías como técnicas de optimización y líneas de productos utilizadas en otras industrias ya consolidadas como alternativas viables para la personalización en masa, reducción de costos y el uso eficiente de los recursos para que las empresas satisfagan los requisitos cambiantes de los clientes y obtengan una ventaja competitiva en los mercados actuales.

Las Líneas de Productos de Software (LPS) son análogas a las líneas de producción industrial, en las cuales se configuran y ensamblan productos similares a partir de la reutilización de insumos y procesos. El objetivo de las LPS es crear la plataforma adecuada para una rápida y fácil producción de software destinado a un mismo segmento de mercado. Dado que estos productos son similares, comparten una serie características comunes y variables. Frecuentemente, las LPS requieren evolucionar o necesitan mantenimiento para satisfacer las necesidades cambiantes del mercado ya sea por funcionalidad, enfoque o tecnología y por consecuencia, es necesario agregar y configurar nuevos productos en la LPS. Sin embargo, los especialistas en el desarrollo de LPS reconocen que son limitadas y que en la mayoría de las ocasiones, es imposible extender o adaptar la plataforma. Una alternativa de solución, es emplear un esquema de reutilización similar a las LPS denominado MPL (Multilínea de Productos de Software).

Una MPL tiene como objetivo la derivación de nuevos productos de software a partir de la reutilización de un conjunto de características provistas por diferentes LPS heterogéneas sin modificar ni alterar el funcionamiento independiente de las mismas.

Esta tesis plantea el desarrollo de un modelo matemático para apoyar la derivación de productos, específicamente en la fase de configuración (selección de características) en una Multilí-

nea de Productos de Software utilizando técnicas de Ingeniería de Software Basada en Búsqueda con el fin de ofrecer una alternativa viable para la toma de decisiones de cómo combinar los insumos de las LPS ya que visualizar las diferentes combinaciones válidas de productos es una tarea compleja y tediosa.

Abstract

Software Engineering incorporates practices or methodologies such as optimization techniques and product lines used in other already consolidated industries as viable alternatives for mass customization, cost reduction and efficient use of resources so that companies meet the changing requirements of customers and gain an advantage competitive in today's markets.

Software Product Lines (SPL) are analogous to industrial production lines, which configured and assembled similar products from the reuse of inputs and processes. The goal of a Software Product Line is to create the right platform for fast and easy production of software for the same market segment. Since these products are similar, they share a number of common and variable features. Software Product Lines often need to evolve or maintenance to meet the changing needs of the market either by functionality, focus or technology and consequently, it is necessary to add and configure new products in the SPL. However, specialists in the development of SPL recognize that they are limited and that in most cases, it is impossible to extend or adapt the platform. An alternative solution is to use a reuse scheme similar to the SPL called MPL (Multi Product Lines). An MPL aims at deriving new software products from the reuse of a set of features provided by different heterogeneous SPLs without modifying or altering the independent operation of the same.

This thesis proposes the development of a mathematical model to support the derivation of products at the stage of an MPL configuration (feature selection) using techniques of Search-Based Software Engineering (SBSE) to offer a viable alternative in making decisions about how to combine the assets of the SPLs since displaying the valid combinations of products, it is a complex and tedious task.

Introducción

Actualmente, los desarrolladores de software recurren al empleo de diversas técnicas, metodologías o prácticas para satisfacer las necesidades de los clientes y cumplir ciertos parámetros de calidad, costos y plazos determinados. Cada vez más, el desarrollo del software avanza hacia la industrialización sustituyendo la forma de desarrollo a la medida por el empleo de enfoques como Líneas de Productos de Software (LPS) que utilizan procesos repetibles, predecibles y controlables para establecer un medio de producción común y ofrecer una variedad de productos que satisfagan las necesidades y requisitos de un segmento de mercado en lugar de enfocarse a un cliente en específico.

La Ingeniería de Líneas de Productos de Software (SPLE) [2, 3] se ocupa de la reutilización sistemática de los activos de desarrollo en un dominio determinado, capturando sus aspectos comunes y variables. Compañías como Siemens, Boeing, HP, Philips, Nokia, Toshiba, Bosch, por mencionar algunas, emplean este paradigma para aumentar su productividad, reducir costos y tiempo de comercialización al lanzar nuevos productos. Sin embargo, la industria del software evoluciona constantemente y requiere atender las necesidades de diferentes segmentos del mercado y diseñar ofertas individuales para cada uno (marketing diferenciado o multi-segmento).

A pesar de que las LPS son de gran utilidad y ofrecen diversos beneficios resultan limitadas e insuficientes para desarrollar sistemas que requieren integrar y reutilizar la funcionalidad de diversos sistemas o componentes independientes y heterogéneos como por ejemplo los Sistemas de Sistemas (SoS) [4] y Ecosistemas de Software (SECO) [5] principalmente empleados en hospitales, sistemas de transporte, edificios inteligentes o plantas industriales. Asimismo, cuando las LPS evolucionan o requieren mantenimiento continuamente para satisfacer las necesidades

cambiantes del mercado ya sea por funcionalidad, enfoque o tecnología minimizando costos y esfuerzo al desarrollar aplicaciones, es necesario emplear un esquema de reutilización similar a las LPS. En los últimos años, han surgido varias propuestas como una solución para el tratamiento de la complejidad, los asuntos de corte (funcionalidad dispersa en diversos dominios) y la variabilidad a través de la composición y reutilización de múltiples LPS, entre las cuales destacan: LPS de LPS, LPS anidadas (Nested Software Product Lines) [6, 7], LPS jerárquicas (Hierarchical Product Lines), LPS compuestas (CPL, Composite Product Lines), conocidas en la literatura como Multilíneas de Productos (MPLs). Una MPL [8, 9, 10, 11] es un enfoque novedoso para desarrollar sistemas grandes y complejos, a través de la reutilización y composición de varias Líneas de Productos de Software independientes, las cuales son desarrolladas por varias organizaciones, con diferentes enfoques y tecnologías con el fin de capturar y gestionar la variabilidad en un dominio complejo. La experiencia en la industria afirma que las MPL se implementan con éxito en otras áreas de Ingeniería como la Siderurgia (*mini mill* [1] de Siemens VAI [12]) y Automotriz. No obstante, la implementación de MPL en el contexto del software se encuentra en desarrollo y las publicaciones que abordan esta área de investigación se encuentran fragmentadas debido a la diversidad de conceptos relacionados y reportan una serie de desafíos (sección 1.1.3.5) al utilizar diversas notaciones de modelado o tecnologías, enfoques, proveedores de componentes de software, equipos de desarrollo, entre otros.

Esta tesis plantea el desarrollo de un modelo matemático para apoyar la derivación de productos específicamente en la fase de configuración (selección de características) en una Multilínea de Productos de Software utilizando técnicas de Ingeniería de Software Basada en Búsqueda debido a que las empresas requieren alternativas para desarrollar o mantener software personalizado que satisfaga las necesidades cambiantes de los clientes a través de técnicas de reutilización, control de la complejidad, separación de asuntos y administración de la variabilidad.

El presente documento está estructurado de la siguiente forma:

[1] Un mini-molino o acería (*mini-mill*) es una fábrica siderúrgica que funde acero reciclado en forma de chatarra para generar productos siderúrgicos básicos).

El capítulo 1 presenta los fundamentos de la investigación actual resaltando los conceptos básicos de las áreas que circunscriben al proyecto, la problemática por la cual surge esta tesis, los objetivos y la justificación del mismo.

El capítulo 2 presenta el estado del arte con el análisis comparativo de los trabajos existentes relacionados con el proyecto actual.

El capítulo 3 describe a detalle la metodología empleada para el desarrollo de la presente investigación.

El capítulo 4 proporciona información acerca de los resultados parciales obtenidos.

Finalmente, el capítulo 5 presenta las conclusiones obtenidas en el desarrollo de este proyecto y el trabajo a futuro.

Capítulo 1

Antecedentes

Este capítulo proporciona la revisión de los diferentes enfoques que apoyan el desarrollo e implementación de Multilíneas de Productos de Software, los conceptos y paradigmas involucrados (Programación Generativa, Líneas de Productos de Software, Multilínea de Productos de Software e Ingeniería de Software Basada en Búsqueda) que son vitales para el correcto entendimiento de la investigación actual. Posteriormente, se presenta el planteamiento del problema, la hipótesis del proyecto, los objetivos general y específicos. Finalmente, se mencionan las aportaciones y la justificación al desarrollo del presente trabajo.

1.1. Marco teórico

La Figura 1.1 describe la revisión de los enfoques para el desarrollo de software los cuales abarcan desde la gestión de la complejidad que implica el software a la medida, la gestión de la diversidad de productos a través de familias o poblaciones, la reutilización y evolución de aspectos comunes y variables a través de Líneas de Productos de Software hasta llegar a la extensibilidad de la plataforma por terceras partes y su integración a través de enfoques como Sistemas de Sistemas (SoS), Ecosistemas de Software (SECOs) y Multilíneas de Productos (MPLs).

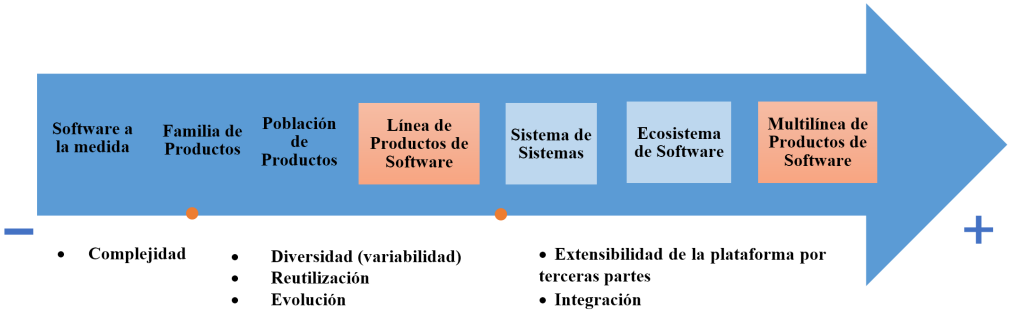


Figura 1.1: Enfoques para el desarrollo de software

El mapa conceptual (Figura 1.2) presenta los principales conceptos involucrados en la literatura de Líneas de Productos de Software (LPS) y Multilíneas de Productos de Software (MPL).

Normalmente, en la literatura se destaca el uso intercambiable de términos como Líneas de Productos de Software, familias de productos y población de productos. Sin embargo, existen pequeñas diferencias entre estos términos que los distinguen en cuestión de alcance. Ommering [13] define una familia de productos como un conjunto de productos que tienen muchas similitudes y pocas diferencias y una población de productos como un conjunto de productos con muchas similitudes, pero también con muchas diferencias, con el desarrollo de partes diseminadas por diferentes sub-organizaciones dentro de una organización más grande.

Con el fin de comprender los enfoques relacionados con MPLs, la Tabla 1.1 presenta las principales características que distinguen cada término.

Una **Línea de Productos de Software (LPS)** es un conjunto de sistemas intensivos de software que comparten una base de código común.

Una **Multilínea de Productos de Software (MPL)** es una composición de LPS interdependientes [14].

Un **Sistema de Sistemas (*SoS, System o System*)** es la composición de sistemas independientes e interoperables destinados a alcanzar objetivos más complejos [4].

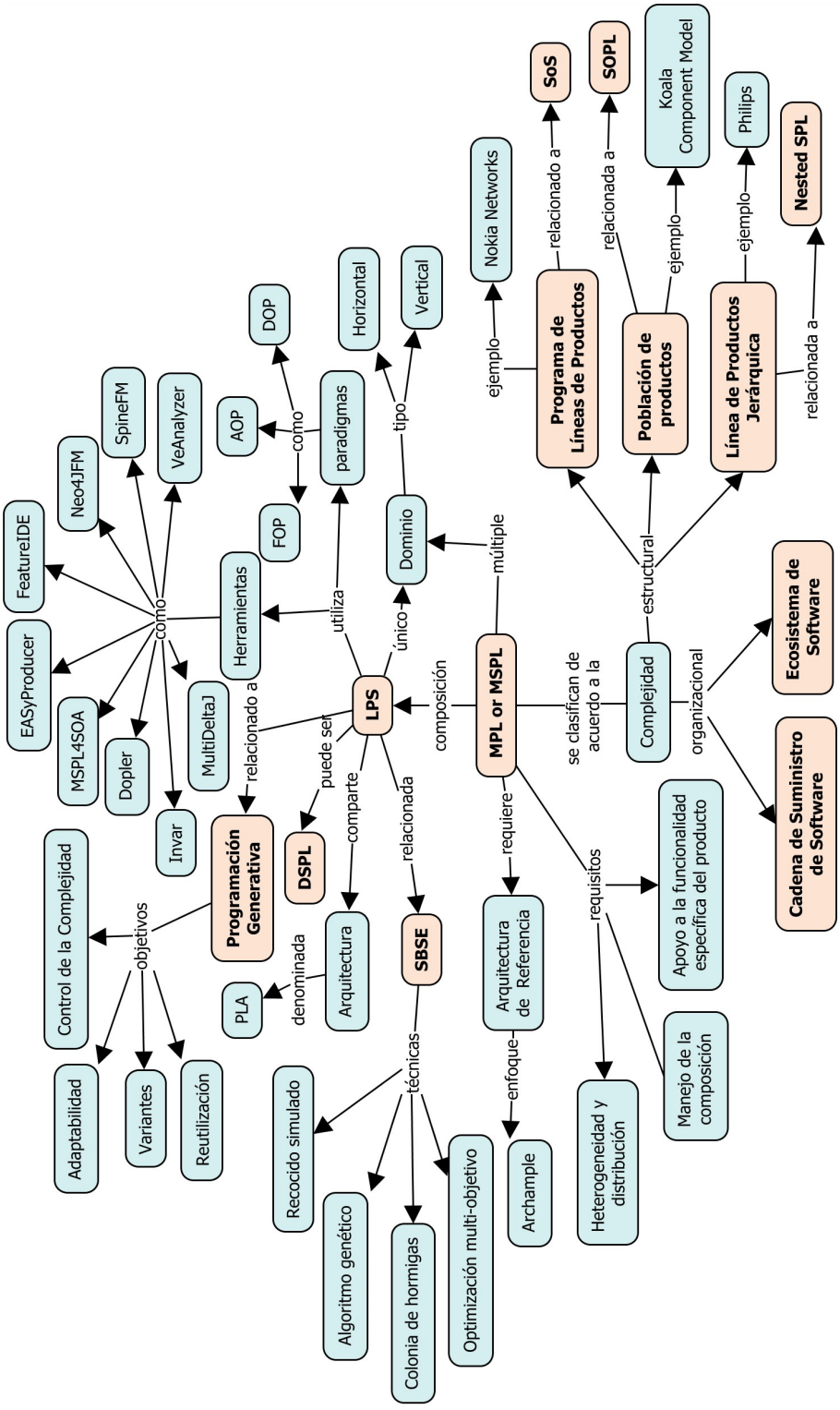


Figura 1.2: Mapa conceptual LPS y MPL

Un **Ecosistema de Software (SECO)** es un conjunto de soluciones de software que permiten, apoyan y automatizan las actividades y transacciones de los actores en el ecosistema social o empresarial asociado y las organizaciones que proporcionan estas soluciones [5]. Un SECO hereda características de los ecosistemas naturales como el mutualismo, el comensalismo, la simbiosis, entre otros [15].

Tabla 1.1: Enfoques relacionados con las MPL

Término	LPS	MPL	SOS	SECO
Relacionado a las LPS	-	Si	Si	Si [5]
Relacionado a las MPL	Si	-	Si	Si [16, 17, 18]
Relacionado a los SoS	Si [4]	Si [19, 20, 21, 22]	-	No
Relacionado a los SECO	Si [5]	Si [17, 23, 24]	No	-
Reutilización de software	Si	Si	No	No
Variabilidad	Un único modelo de características	Múltiples modelos de características	No aplica	Amplio rango de configuraciones [15]. Ecosistemas enriquecidos con variabilidad [25]
Dominio	Único	Multiplicidad	Único	Multiplicidad
Caraterísticas		Heterogeneidad de las LPS involucradas.	Independencia operacional y gerencial, desarrollo evolutivo, comportamiento emergente y distribución geográfica. [26]	Estabilidad, gestión de la evolución, seguridad y confiabilidad.

Continúa en la página siguiente

Tabla 1.1: Enfoques relacionados con las MPL (continuación)

Término	LPS	MPL	SOS	SECO
Activos base	Componentes, programas, biblioteca, artefactos, PLA.	Componentes, programas, blibliotecas, artefactos, PLA, sistemas, LPS, Arquitectura de Referencia.	No aplica	No aplica
Productos	Componentes, programas, bibliotecas, artefactos, sistemas.	Componentes, programas, bibliotecas, artefactos, sistemas y LPS.	No aplica	No aplica

LPS, MPL, SoS y SECO son similares debido a que cada término gestiona un conjunto de sistemas de software estrechamente relacionados. Del mismo modo, SECOs y LPS son enfoques que agrupan activos variables alrededor de una plataforma tecnológica común. Un SECO es un tipo de evolución natural de las LPS. Los enfoques de MPL y SECO tratan con múltiples proveedores que contribuyen a un modelo compartido de variabilidad. En ambos enfoques, se destaca la participación de una comunidad externa de desarrolladores y expertos que extienden una plataforma de software para satisfacer las necesidades de dominios específicos. Por lo tanto, un SECO se observa como una MPL donde el espacio variante está abierto. Además, las LPS son factibles de utilizarse para construir componentes de un SoS. Por otro lado, en un SoS es posible identificar la coexistencia y la colaboración de múltiples sistemas. En consecuencia, es posible la coexistencia de una MPL que soporta la implementación de nuevas características de un SoS.

1.1.1. Programación Generativa

La Programación Generativa es útil para implementar Líneas de Productos de Software ya que es posible derivar automáticamente cada producto basándose en una configuración adecuada. El objetivo principal de la Programación Generativa es maximizar la automatización del desarrollo de aplicaciones a partir de su especificación.

El desarrollo de software generativo [27, 28] es un enfoque de familias de sistemas, el cual se centra en la automatización de miembros de la familia de sistemas: un sistema dado se genera automáticamente a partir de una especificación descrita en un lenguaje de dominio específico (textos o gráficos).

La **Programación Generativa (*Generative Programming*)** es un paradigma para el desarrollo de software que fusiona la Ingeniería del Dominio y las técnicas orientadas a objetos para lograr alta intencionalidad, reutilización y adaptabilidad sin la necesidad de comprometer el desempeño de ejecución.

Czarnecki [27] define a la **Programación Generativa** como un paradigma de Ingeniería de Software basado en el modelado de familias de software que a partir de una especificación de requisitos, produce un producto intermedio o final altamente personalizado y optimizado a partir de componentes elementales y reutilizables mediante conocimiento de configuración.

1.1.1.1. Objetivos de la Programación Generativa

Los objetivos de la Programación Generativa son:

- Incrementar la reutilización, adaptabilidad y eficiencia (espacio y tiempo de ejecución).
- Mejorar el control de la complejidad.
- Administrar un gran número de variantes.

Estos objetivos se logran mediante la aplicación de los siguientes principios fundamentales:

- **Separación de asuntos (*SOC*):** para la realización del software, es necesario identificar cada asunto, separarse y posiblemente encapsularse.

- **Apertura de la implementación del modelo:** cada componente proporciona acceso a sus estrategias de implementación como por ejemplo los aspectos.

- **Proporcionar aspectos alternativos de un asunto:** un aspecto es un componente que implementa un asunto de otro componente. Los asuntos relacionados de un componente se modelan como sus parámetros.

- **Propagación de aspectos:** los aspectos suelen pasar de un componente a otro. Los aspectos se propagan de sus contenedores a sus partes y de sus partes a sus contenedores. La propagación de los aspectos extiende su alcance a un conjunto de componentes. Se minimiza la redundancia y se facilitan las optimizaciones específicas del dominio global.

- **Ocultar la complejidad utilizando capas y reglas de configuración:** los componentes tienen algunos aspectos externos (características) y por lo general una gran número de aspectos internos. Los aspectos internos se derivan automáticamente de los aspectos externos utilizando reglas de configuración para expresar restricciones.

1.1.1.2. Generadores

Un generador es un programa que transforma una especificación de alto nivel en una de bajo nivel, por ejemplo: compiladores (rmic en Java), CORBA IDL, por mencionar algunos. Actualmente, casi todas las herramientas UML o GUI Builder proveen un generador de código automático.

Czarnecki [29] define a un generador como un programa que dada una especificación de alto nivel produce su implementación. La pieza de software es un sistema, un componente, una clase o un procedimiento.

La diferencia de un compilador y un generador de programas, es que que el generador produce un programa escrito en lenguaje de alto nivel en vez de código máquina. Sin embargo, el concepto de generador va mucho más allá, ya que permite:

- **Expresar intencionalmente la especificación del sistema:** las descripciones intencionales son fáciles de entender, analizar, verificar y mantener, es decir, poseen todas las buenas cualidades de software. Se expresan en un DSL y se implementan con un generador.
- **Mover la verificación de nivel de código a la especificación intencional:** el código producido por un generador es correcto por la construcción, al menos parte de la verificación se mueve de nivel de código a nivel de especificación disminuyendo el costo de esta fase.
- **Calcular una implementación eficiente:** los generadores mapean la especificación intencional de forma eficiente ya que facilitan las transformaciones. El mapeo de una especificación de nivel superior a un nivel inferior se denomina transformación. En realidad, el desarrollo de software es una secuencia de transformaciones y modificación de las especificaciones.
- **Factorización (especialización) de código:** dado que los generadores operan en un conjunto definido de pequeños componentes configurables, el código resultante es altamente factorizado, por lo tanto, es fácil de entender, mantener y reutilizar.

1.1.1.3. Técnicas de la Programación Generativa

Las técnicas para desarrollar generadores se clasifican en:

1. **Ad-hoc (específico)**: los generadores se desarrollan utilizando técnicas y herramientas de implementación para compiladores. La mayoría de su implementación se escribe en un lenguaje de propósito general y el generador es un ejecutable independiente. Ejemplo: compiladores y generadores de analizadores sintácticos como *Lex* y *Yacc*.
2. **Metaprogramación**: es posible construir generadores utilizando bibliotecas de metaprogramación o lenguajes de programación reflexivo. Ejemplo: *Smalltalk* y *OpenC++*.

3. **Transformacional**: es posible implementar un generador como un conjunto de reglas de transformación que se aplican a una representación interna de la especificación de entrada. Ejemplo: *Draco* y *ASF+SDF* [1].

4. **Composicional**: los programas se generan por la unión de pequeñas partes de programas. Estos generadores se implementan utilizando composición de sistemas, el cual se compone de fragmentos de programas genéricos. Estos fragmentos de programas se parametrizan de forma que se personalizan en una composición particular. Genera un programa de salida mediante la selección de los fragmentos de programa apropiados, basados en la especificación de entrada y los ensambla para producir el programa. Ejemplo: *GenVoca*.

1.1.1.4. Paradigmas relacionados con la Programación Generativa

Existen otros paradigmas que tienen objetivos similares a la Programación Generativa como [29]:

- **Programación Genérica (*Generic programming*):** se refiere a la reutilización a través de la parametrización. Permite que los componentes ampliamente personalizables, conserven la eficiencia del código configurado estáticamente.

- **DSL (*Domain-Specific Languages*, Lenguajes de Dominio Específico):** lenguaje de programación o especificación de un lenguaje dedicado a resolver o representar un problema en particular (dominio).

- **AOP (*Aspect-Oriented Programming*, Programación Orientada a Aspectos):** técnica de programación que provee al programador de mecanismos adecuados para separar componentes y aspectos, unos de otros, permitiendo su abstracción y composición para producir todo el sistema [30].

[1] IDE (Entorno de Desarrollo Integrado) y conjunto de herramientas para el análisis y transformación de programas interactivos. Combina SDF (Syntax Definition Formalism), ASF (Algebraic Specification Formalism) y otras tecnologías.

La relación entre el Desarrollo de Software Generativo y otras áreas emergentes [28] se observa en la figura 1.3:

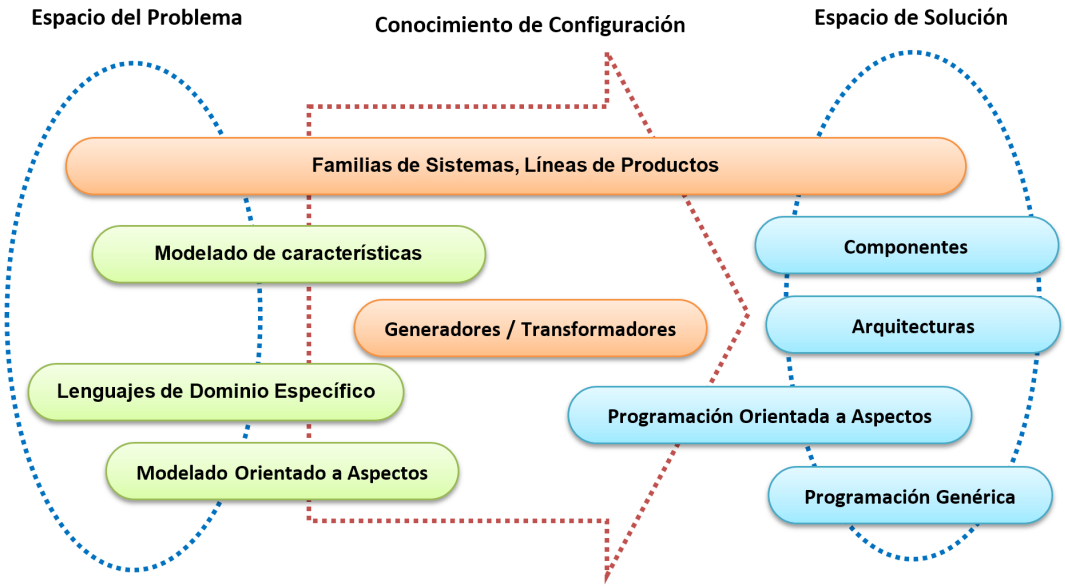


Figura 1.3: Relación del DSG y otras áreas emergentes

1. Componentes, arquitecturas y la Programación Genérica se relacionan principalmente con el espacio de solución.

2. La Programación Orientada a Aspectos proporciona mecanismos de encapsulación de asuntos de corte, por lo tanto cubre el espacio de solución y una parte del conocimiento de configuración. Los aspectos se localizan en el espacio del problema especialmente en el contexto de los DSL.

3. Las familias de sistemas y la Ingeniería de las Líneas de Productos de Software abarcan todo el modelo de dominio generativo ya que proporcionan la estructura general del proceso de desarrollo (Ingeniería del Dominio e Ingeniería de Aplicación).

1.1.2. Línea de Productos de Software

El concepto de Línea de Productos de Software se utiliza tanto en la industria como en la academia para referirse al desarrollo de un conjunto de productos de software similares utilizando una plataforma y personalización en masa.

El Instituto de Ingeniería de Software de la Universidad Carnegie Mellon (SEI) [2] define una **Línea de Productos de Software** (*SPL, Software Product Line*) como: "un conjunto de sistemas de software intensivo que comparten una serie de características administradas que satisfacen las necesidades específicas de un segmento de mercado particular o misión, y que se desarrollan de forma prescrita a partir de un conjunto común de activos base (*core assets*)". Algunos de los beneficios de las LPS son:

- Aumento de la productividad a gran escala.
- Mejora la calidad de los productos de software.
- Reducción del esfuerzo de mantenimiento.
- Personalización en masa.
- Reducción de los costos de desarrollo.
- Reducción del tiempo al tomar la decisión de desarrollar un producto y la fecha del lanzamiento comercial (*time to market*).
- Reducción de los riesgos.
- Crecimiento de la organización.
- Incremento en la satisfacción del cliente.
- Mantener la presencia en el mercado.

1.1.2.1. Terminología de las LPS

A continuación, se describe la terminología asociada con las Líneas de Productos de Software:

- **Activos base (*core assets*)**: artefactos y recursos reutilizables que forman las bases para la Línea de Productos de Software. Incluyen la arquitectura, componentes de software reutilizables, modelos de dominio, requisitos, documentación, especificaciones, calendario, presupuesto, planes de prueba, casos de prueba, por mencionar algunos.

- **Alcance (*scope*):** actividad que limita un sistema o conjunto de sistemas mediante la definición de aquellos comportamientos o aspectos que se encuentran dentro y fuera. El alcance define la colección de artefactos de software que constituyen la Línea de Productos de Software. Es decir, identifica el intervalo (alcance) de características que los sistemas de la LPS cubren. Según Schmid [31], se distinguen tres tipos de alcance:

  - **Alcance de la cartera de productos** tiene como objetivo identificar los productos particulares a desarrollar, así como las características que proporcionan.
  - **Alcance de dominio** es la tarea de limitar los dominios que son relevantes para la LPS.
  - **Alcance de los activos** tiene como objetivo identificar las partes funcionales de la LPS a desarrollar de forma reutilizable.

- **Arquitectura:** "conjunto de estructuras necesarias para razonar sobre el sistema, el cual comprende elementos de software, relaciones entre ellos, y las propiedades de ambos. Las estructuras consisten de elementos, relaciones entre elementos y las propiedades importantes de ambos. Así que documentar una estructura implica documentar esos aspectos [32] ".

- **Arquitectura de la Línea de Productos (*PLA, Product Line Architecture*):** representa la arquitectura de software genérica común para los productos de la Línea de productos.

- **Aspectos comunes (*commonalities*):** denotan las características que están presentes de la misma forma en todas las aplicaciones.

- **Característica *(feature)***: rasgo o elemento distintivo percibido por el usuario final que representa aspectos relevantes del software. Se utiliza para describir o distinguir un producto de la LPS.

- **Configuración:** proceso de toma de decisiones en la selección de variantes, combinación de componentes, opciones asociadas y ajustes que un producto de software implementa de forma parcial o completa.

- **Componente:** es una unidad de composición con especificaciones explícitas de interfaces requeridas y provistas y atributos de calidad.

- **Derivación de un producto:** proceso completo de construir un producto a través de los activos de software de un familia de productos.

- **Dimensión de la variabilidad:** tipo de variabilidad que es importante para un stakeholder, por ejemplo el dominio de la aplicación, el entorno de ejecución de un programa, el contexto en tiempo de ejecución, las propiedades no funcionales y la implementación de la variabilidad [33].

- **Dominio:** área de conocimiento o actividad caracterizada por un conjunto de conceptos y terminología comprendida por los practicantes en esa área.

- **Modelo de características *(Feature Model)*:** técnica para expresar variabilidad a través de las características (obligatorias, opcionales y alternativas) de un producto.

- **Selección de características:** obtención de una instancia del modelo de características, la cual representa las características de una aplicación.

- **Punto de variación:** es una decisión de diseño que identifica el lugar en el que los productos varían entre sí, es decir lo que varía en un contexto dado.

- **Variabilidad:** se refiere a la capacidad de un artefacto o sistema de software para configurarse, personalizarse, extenderse o cambiarse para su uso en un contexto específico.

- **Variante:** representación de un objeto de variación dentro de los artefactos de dominio (un posible valor para un punto de variación).

1.1.2.2. Procesos de desarrollo

El paradigma de la Ingeniería de Líneas de Productos de Software separa dos procesos (Figura 1.4) [3]:

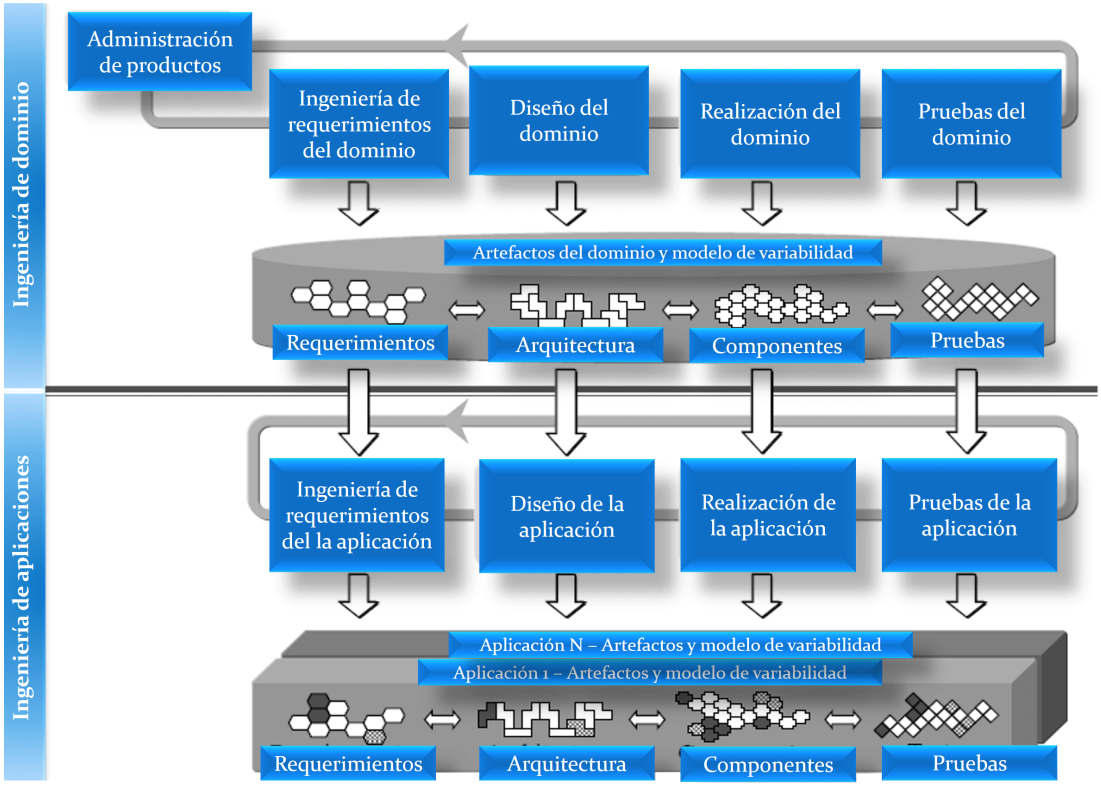


Figura 1.4: Procesos LPS

1. **Ingeniería de Dominio:** proceso responsable de establecer la plataforma reutilizable, definir lo que es común y lo que es variante de un producto de la línea. La plataforma consiste de todos los tipos de artefactos de software (requisitos, diseño, realización, pruebas, en-

tre otros). La conexión de trazabilidad entre estos artefactos facilita sistemáticamente y consistentemente la reutilización. Los sub-procesos de la Ingeniería del Dominio son:

- **Administración del producto:** se refiere al proceso de controlar el desarrollo, producción y mercadeo de la LPS y sus productos.
- **Ingeniería de requerimientos del dominio**: comprende las actividades para identificar y documentar los requerimientos comunes y variables de la LPS.
- **Diseño del dominio**: comprende las actividades para definir la arquitectura de referencia de la LPS, la cual provee una estructura común y de alto nivel para todas las aplicaciones.
- **Implementación del dominio**: comprende el diseño detallado y la implementacion de componentes reutilizables. Los resultados son componentes configurables y débilmente acoplados (no una aplicación en ejecución).
- **Prueba del dominio**: responsable de validar y verificar los componentes reutilizables. Se encarga de desarrollar artefactos de prueba reutilizables para reducir el esfuerzo de prueba de la aplicacion.

2. **Ingeniería de Aplicación:** proceso en el que las aplicaciones de la línea de productos se construyen mediante la reutilización de artefactos del dominio y la explotación de la variabilidad de la línea de productos. Este proceso se conoce también como **derivación del producto** [34]. Los sub-procesos de la Ingeniería de Aplicación son:

   - **Ingeniería de requerimientos de la aplicación**: comprende todas las actividades para desarrollar la especificación de requerimientos de la aplicación.
   - **Diseño de la aplicación:** comprende las actividades para producir la arquitectura de la aplicación. El diseño de la aplicación utiliza la arquitectura de referencia para instanciar la arquitectura de la aplicación. La arquitectura de la aplicación se deriva de la arquitectura de referencia enlazando variabilidad.

- **Implementación de la aplicación**: se encarga de crear la aplicación según la selección y configuración de componentes de software reutilizables así como la implementación de insumos específicos de la aplicación.
- **Prueba de la aplicación**: comprende las actividades necesarias para validar y verificar una aplicación con su especificación.

Los artefactos de desarrollo se refieren a la salida de un sub-proceso de la ingeniería del dominio o de la aplicación (requerimientos, arquitectura, componentes, pruebas). El marco de trabajo diferencia entre diferentes tipos de artefactos de desarrollo:

- Los artefactos de dominio incluyen la plataforma de la línea de productos de software.
- Los artefactos de aplicación representan todo tipo de artefactos de desarrollo de aplicaciones específicas.

1.1.2.3. Enfoques para desarrollar LPS

Existen tres enfoques diferentes como estrategia para realizar la adopción de una Línea de Productos de Software (Figura 1.5):

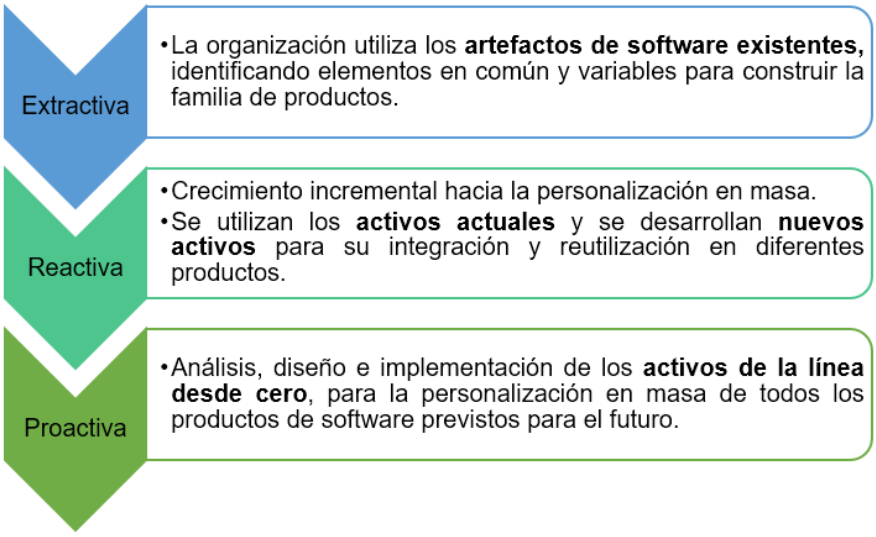


Figura 1.5: Estrategias de desarrollo de LPS

1.1.2.4. Principios de Variabilidad

La gestión de la variabilidad es lo que diferencia la Ingeniería de Líneas de Productos de Software de la Ingeniería de Software tradicional. El proceso de desarrollo de una LPS implica gestionar los puntos de variación entre los diferentes miembros de la línea. Por esta razón se identifican los aspectos comunes y variables del dominio en cuestión.

El propósito general de la variabilidad de una LPS es maximizar el retorno de la inversión (*ROI, Return On Investment*) para construir y mantener productos durante un período específico de tiempo.

Los aspectos comunes (*commonalities*) denotan características que están presentes en todas las aplicaciones en exactamente de la misma forma.

Las características variables son aquellas que varían de un sistema a otro. La variabilidad de una LPS se define en el proceso de Ingeniería del Dominio mediante un modelo de variabilidad que captura la variabilidad del dominio en requisitos, arquitectura, componentes y pruebas, y se explota en la etapa de Ingeniería de Aplicación.

A continuación, se definen algunos conceptos básicos relacionados con el concepto de variabilidad [3]:

- **Sujeto de variabilidad (*variability subject*):** elemento variable del mundo real o una propiedad variable del propio elemento. Ejemplo: forma de pago.
- **Objeto de variabilidad (*variability object*):** instancia particular de un sujeto de variabilidad. Ejemplo: tarjetas (crédito, debito) o efectivo.
- **Punto de variación (*variation point*):** representación de un sujeto de variabilidad dentro de los artefactos de dominio enriquecido por información contextual. Es decir, lo que varía en un contexto dado.
- **Variante (*variant*):** representación de un objeto de variación dentro de los artefactos de dominio. Es decir, un posible valor para un punto de variación.

Los puntos de variabilidad y variantes se utilizan para definir la variabilidad de la LPS. El proceso para identificar la variabilidad es:

1. Identificar los elementos en el mundo real que varían (Sujeto de variabilidad).
2. Definir los puntos de variación dentro del contexto de la LPS.
3. Definir las variantes.

Según Pohl [3] los diferentes tipos de variabilidad son:

- **En el tiempo:** es la existencia de diferentes versiones de un artefacto que son válidos en diferentes momentos.
- **En el espacio:** es la existencia de un artefacto en diferentes formas al mismo tiempo.
- **Externa:** es la variabilidad visible para los clientes de los artefactos de dominio.
- **Interna:** es la variabilidad que se oculta a los clientes de los artefactos de dominio.

1.1.2.5. Modelos de características

Una **característica (*feature*)** se define como un elemento visible del sistema, de interés para alguna persona que interactúa con el sistema, ya sea un usuario final o un desarrollador software. Una característica se relaciona con las funcionalidades del software o atributos del sistema visibles para el usuario. Las características permiten distinguir los productos y son importantes para representar la variabilidad [35, 36]. En este sentido, es posible generar diferentes productos seleccionando diferentes características.

Un **modelo de características (*FM, Feature Model*)** representa de forma gráfica o textual a una LPS a través de las posibles combinaciones entre las características. Se utilizan para manejar la variabilidad y representar el conjunto de productos en una LPS. Estos productos se definen como un conjunto de características, cada una de las cuales describe un incremento en la funcionalidad del producto.

Los modelos de características codifican la variabilidad de una LPS en términos de características obligatorias, opcionales y exclusivas, así como restricciones proposicionales sobre las

características. Los modelos de características delimitan el alcance de la LPS y documentan formalmente las configuraciones que son soportadas. Una vez especificados, es posible utilizar los modelos de características para el modelo de verificación de una LPS, para probar la LPS, para la automatización de la configuración del producto, o para el cálculo de la información pertinente.

En los modelos de características, el número de productos potenciales de una LPS aumenta con el número de características. En consecuencia, un gran número de características conducen a una LPS con un gran número de productos potenciales. En una LPS flexible, es posible que todas las características aparezcan o no en todos los productos potenciales. El conjunto de características que componen un modelo de características se organizan de la siguiente forma:

- **Relaciones jerárquicas:** relaciones entre una característica padre y sus características hijas (sub-características).
  - **Obligatoria (*Mandatory*):** indica que la característica requiere seleccionar la sub-característica.
  - **Opcional (*Optional*):** indica la posibilidad de seleccionar sub-características opcionales.
  - **Alternativas (*Alternative*:)** solo es posible seleccionar una de las sub-características del grupo.
  - ***Or:*** indica que se elige una o más sub-características en el grupo. Es posible colocar una restricción de cardinalidad para restringir el número mínimo y máximo de características seleccionables en un grupo.
- **Relaciones no jerárquicas:** suelen ser del tipo si la característica A aparece, entonces la característica B se incluye o excluye.
  - **Require *(Requires):*** si una característica G requiere una característica C significa que si la característica G es incluida en el producto, es necesario incluir la característica C, pero no viceversa.
  - **Excluye (*Excludes*):** si una característica H excluye una característica C significa que ambas características no forman parte del mismo producto.

La figura 1.6 describe las posibles relaciones entre las características.

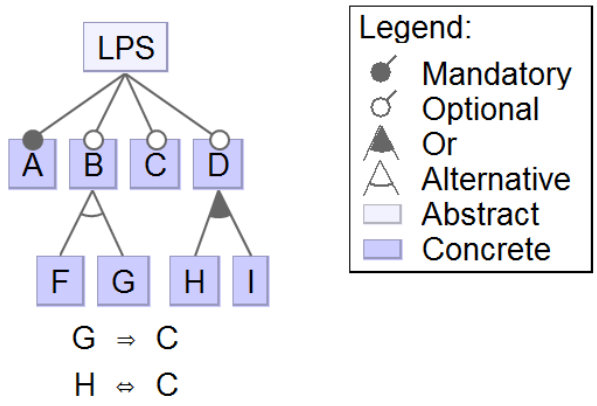


Figura 1.6: Modelo de características en FeatureIDE

1.1.2.6. Análisis de los modelos de características

El gran número de configuraciones que un modelo de características codifica hace que el análisis manual sea una tarea propensa a errores y costosa. Los mecanismos asistidos por computadora aparecieron como una solución para guiar y ayudar a los profesionales en diferentes tareas de la Ingeniería del Software como depuración, configuración o pruebas. Este proceso se conoce como **análisis automático de modelos de características** [35, 36]. Este concepto es clave cuando se razonan sobre los sistemas de gran variabilidad y de gran complejidad. El objetivo del análisis automático es auxiliar a los gestores, analistas y diseñadores en el desarrollo de la LPS.

El análisis de los modelos de características se realiza en términos de operaciones de análisis. Una operación toma un conjunto de parámetros como entrada y devuelve un resultado como salida. Además de los modelos de características, los parámetros típicos de entrada y salida son:

- **Configuración:** dado un modelo de características con un conjunto de caracteristicas, una configuración es una 2-tupla de la forma (S, R) tal que $S, R \subseteq F$ donde S es el conjunto de características a ser seleccionadas y R el conjunto de características a removerse tal que $S \cap R = 0$.

- **Configuración completa:** si la configuración $S \cup R = F$.

  Si se tiene el modelo de características de la *LPS MobilePhone* (figura 1.7), un ejemplo de configuración completa es:

  FC=({MobilePhone, Calls, Screen, Color }, {GPS, Basic, HighResolution, Media, Camera, MP3})

- **Configuración pacial**: si la configuración $S \cup R \subset F$.

  Si se tiene el modelo de características de la *LPS MobilePhone* (figura 1.7), un ejemplo de configuración parcial es:

  PC=({MobilePhone, Calls, Camera }, {GPS})

- **Producto:** un producto es equivalente a una configuración completa en la que sólo se especifican las características seleccionadas y las características omitidas se eliminan implícitamente. Por ejemplo, el siguiente producto es equivalente a la configuración completa descrita anteriormente:

  P = {MobilePhone, Calls, Screen, Color}

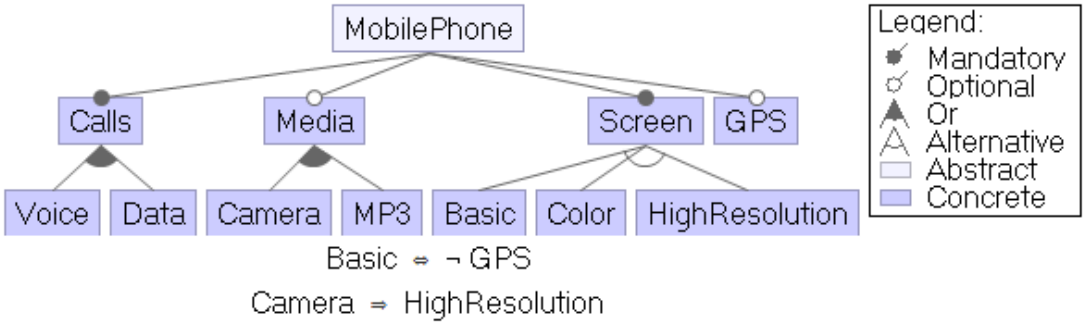


Figura 1.7: Modelo de características de la LPS MobilePhone

Existen varias operaciones y diversas propuestas [35, 36] para automatizar estas operaciones, entre las que destacan: .

- **Determinar si un producto es válido para una LPS.** Esta operación toma como entrada un modelo y un producto (un conjunto de características) y retorna un valor determinando

si el producto pertenece al modelo o no. Por ejemplo, si se tiene los productos P1 y P2 y el modelo de características parcial de la *LPS MobilePhone* (figura 1.8). El producto *P1* no es un producto válido para el modelo porque no incluye la característica obligatoria *Calls*, mientras que el producto *P2* es un producto válido para el modelo de características.

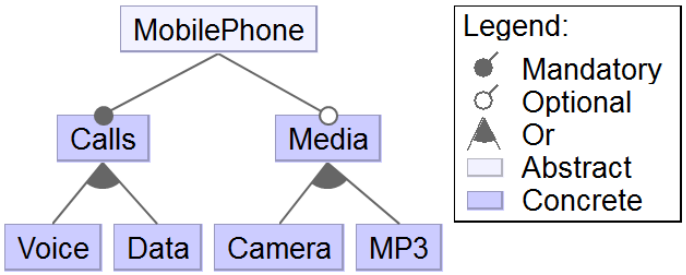


Figura 1.8: Modelo de características parcial de la LPS MobilePhone

P1 = {MobilePhone, Media, MP3, Camera}

P2 = {MobilePhone, Calls, Voice, Media, MP3}

- **Determinar si un modelo es válido.** Esta operación toma como entrada un modelo y retorna un valor determinando si tal modelo es válido o no. Un modelo es válido cuando representa por lo menos un producto. Un modelo de características es inválido cuando se utilizan de forma incorrecta las relaciones no jerárquicas entre las características, generando de esta manera contradicciones en el modelo.

- **Obtener todos los posibles productos.** Esta operación toma como entrada un modelo y retorna todos los productos válidos representados por dicho modelo.

- **Obtener todos los posibles productos.** Esta operación toma como entrada un modelo y retorna todos los productos válidos representados por dicho modelo. Un ejemplo de esta operación aplicada al modelo de la figura 1.8 resulta en el siguiente conjunto de productos:

  P1 = {MobilePhone, Calls, Voice}

  P2 = {MobilePhone, Calls, Voice, Media, Camera}

  P3 = {MobilePhone, Calls, Voice, Media, MP3}

P4 = {MobilePhone, Calls, Voice, Media, Camera, MP3}

P5 = {MobilePhone, Calls, Data}

P6 = {MobilePhone, Calls, Data, Media, Camera}

P7 = {MobilePhone, Calls, Data, Media, MP3}

P8 = {MobilePhone, Calls, Data, Media, Camera, MP3}

P9 = {MobilePhone, Calls, Voice, Data}

P10 = {MobilePhone, Calls, Voice, Data, Media, Camera}

P11 = {MobilePhone, Calls, Voice, Data, Media, MP3}

P12 = {MobilePhone, Calls, Voice, Data, Media, Camera, MP3}

- **Determinar si dos modelos son equivalentes.** Esta operación toma como entrada dos modelos y retorna un valor determinando si los modelos son equivalentes. Dos modelos de características son equivalentes si representan el mismo conjunto de productos. Benavides clasifica los modelos equivalentes en dos tipos:

  1. **Totalmente equivalentes.** Dos modelos son totalmente equivalentes si ellos representan el mismo conjunto de productos y el mismo conjunto de características.
  2. **Parcialmente equivalentes.** Dos modelos son parcialmente equivalentes si representan el mismo conjunto de productos pero no necesariamente usan el mismo conjunto de características.

- **Obtener las características básicas (*core features*).** Esta operación toma como entrada un modelo y retorna el conjunto de características que aparecen en todos los productos. Por ejemplo, si consideramos el modelo de características (figura 1.8), el conjunto de características común a todos los productos es {MobilePhone, Calls}.

- **Obtener las características variantes (*variant features*).** Esta operación toma como entrada un modelo y retorna el conjunto de características que no aparecen en todos los productos. Por ejemplo, si consideramos el modelo de características (figura 1.8), el conjunto de características común a todos los productos es {Voice, Data, Media, Camera, MP3}.

- **Calcular el número de productos.** Esta operación toma como entrada un modelo y retorna el número de productos que este representa. Esta operación ofrece información respecto a la complejidad de la LPS, ya que una LPS con una gran cantidad de productos exige más esfuerzo de gestión y desarrollo. Por ejemplo, si consideramos el modelo de características (figura 1.8) representan 12 productos distintos.

- **Calcular el número de productos potenciales.** Esta operación toma como entrada un modelo y retorna la proporción entre el número de productos de un modelo y el número de productos potenciales si todas las características se combinaran entre ellas sin restricciones. El número de productos potenciales se representa por $2^n - 1$ donde n es el número de características a considerar. Generalmente, solo se consideran las características hojas (excepto la característica raíz). Por ejemplo, si consideramos el modelo de características (figura 1.8), entonces:

$$\frac{Número \quad de \quad productos}{2^n - 1} = \frac{12}{2^6 - 1} = 0{,}19047 \tag{1.1}$$

  Esta operación es útil para medir la flexibilidad de la LPS, es decir, un valor pequeño en su resultado significa que el número de combinaciones posibles de las características del modelo es limitado si se compara al número total de productos potenciales.

- **Filtrar un conjunto de productos.** Esta operación toma como entrada un modelo, un conjunto de características F_i a ser incluido y un conjunto de características F_e a ser excluido y retorna el conjunto de productos que poseen las características de F_i y no poseen las características de F_e. Algunas propuestas encontradas en la literatura estudian la aplicación de filtro a los modelos de características. Esta operación es importante en el contexto de la Ingeniería de Aplicación, donde es necesario elegir las características para cada producto específico.

- **Calcular la similaridad (*commonality*).** Esta operación toma como entrada un modelo y una característica del modelo y retorna un valor que representa el porcentaje de productos válidos en los que aparece la característica.

- **Detectar características muertas (*dead feature*s).** Esta operación toma como entrada un modelo de características y retorna el conjunto de características muertas, es decir aquéllas que no aparecen en ningún producto de la LPS. Las características muertas se originan cuando se utilizan incorrectamente las relaciones no jerárquicas. Se requiere que el modelo de características sea válido y no posea características muertas.

1.1.2.7. Problemas de configuración en las LPS

Principalmente, los problemas a enfrentar durante la configuración de las Líneas de Productos de Software se enumeran a continuación [37]:

- **Modelo de características vacío (*Void feature model*):** un modelo de características es nulo si no hay algún producto válido en la derivación. Es no nulo si tiene al menos un producto válido.
- **Modelo de características inválido (*Invalid feature model*):** un modelo de características es no válido si sólo produce un producto.
- **Producto inconsistente (conflictos de características):** un producto es inconsistente si infringe las reglas y regulaciones del modelo de características.
- **Modelo de características inconsistente:** un modelo de características es inconsistente si sus características incluidas no son consistentes entre sí.
- **Problema de la derivación (configuración) del producto:** es posible derivar múltiples productos de un único dominio en el punto de variación.
- **Problema de la gestión de la variabilidad:** la gestión de la variabilidad se ocupa de las diferentes variaciones en la implementación del software.
- **Errores de características:** otros errores relacionados con los modelos de características son: características redundantes, cardinalidades inadecuadas y características variables muertas y falsas. Un modelo de características es redundante si la información semántica

(al menos una) es modelada de una manera múltiple. Una cardinalidad inapropiada adopta más características clonadas que las permitidas por el modelo de características. Las características muertas se definen en los modelos de características, pero nunca son parte de una configuración de producto válida. Las características de las variables falsas son aquellas que se etiquetan como opcionales en los modelos de características, pero tienen una restricción de selección obligatoria en el caso de la selección de características de los padres.

1.1.3. Multilínea de Productos de Software

Varios autores [38, 39, 40, 41, 10, 42, 43] han discutido la necesidad de utilizar múltiples líneas de productos para la reutilización de artefactos de requisitos para diferentes Líneas de Productos de Software.

Una **Multilínea de Productos** (*MPL, Multi Product Line*) también denominada **MultiLínea de Productos de Software** (MSPL, Multiple Software Product Line) es un tipo especial de Línea de Productos de Software que extiende la reutilización de software mediante la administración de varias Líneas de Productos de Software heterogéneas, es decir, que se desarrollan, administran y despliegan de forma independiente. El enfoque se refiere al desarrollo de software en el que los productos de software son el resultado de combinar componentes o productos desarrollados en Líneas de Productos de Software independientes, las cúales provienen de diversas organizaciones o equipos, utilizan diversos enfoques, tecnologías y proveedores. Los términos MPL, MSPL, Múltiples LPS, Líneas de Productos Anidadas, Líneas de Productos de Líneas de Productos se utilizan para denotar el mismo concepto.

1.1.3.1. Definición de MPL

Acher [8, 41] define una **MPL** como “una LPS que gestiona un conjunto de Líneas de Productos de Software $\{LPS_1, LPS_2, ... LPS_n\}$. El conjunto de productos se describe mediante un modelo de características FM_{MPL} ”. De acuerdo con Holl [10], una MPL es un conjunto de va-

rias Líneas de Productos independientes (autónomas) que en conjunto representan un sistema a gran escala o ultraescala.

Otro término relacionado con la MPLs es la **Línea de Productos de Software Anidada** *(Nested SPL, Nested Software Product Line)* que representa una composición jerárquica de grandes LPS a partir de pequeñas LPS con el fin de reducir la complejidad de los modelos de características [6, 7]. MPL y LPS anidadas se ocupan de los asuntos de corte *(cross-cutting concerns)*, es decir, asuntos que interfieren con la funcionalidad de otros o requisitos que se encuentran dispersos en diferentes LPS y por consecuencia se encuentran presentes en diversos dominios (horizontal o vertical).

Para ilustrar la composición recursiva de LPS se recurre a la aplicación del patrón de diseño *composite*. Es posible construir un sistema a gran escala y complejo utilizando varias LPS, las cuales son planas, anidadas o **Líneas de Productos Compuestas** *CPL, Composite Product Line* que resultan de la composición de LPS. La Figura 1.9 presenta el modelo conceptual propuesto en [44, 45] para el desarrollo de sistemas utilizando MPLs.

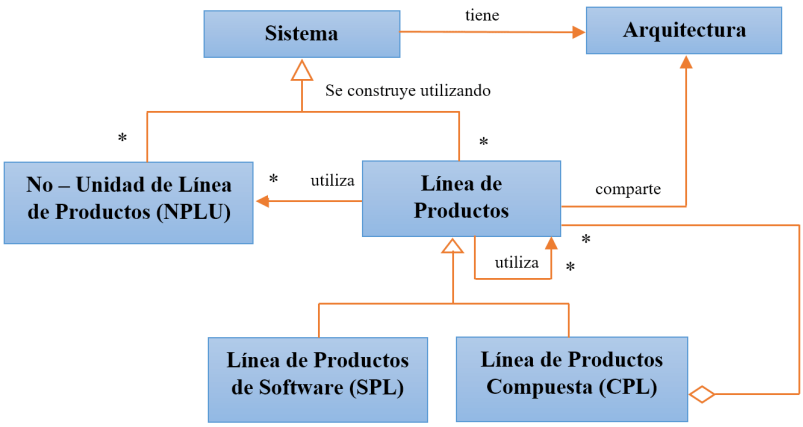


Figura 1.9: Modelo conceptual para el desarrollo de sistemas utilizando MPLs

MPL Competitiva y MPL Composicional son otros términos involucrados en las MPLs [8]. La Tabla 1.2 presenta las diferencias entre MPL Competitivas y MPL Composicionales.

En una **MPL Competitiva**, cada LPS generalmente proporciona diferentes conjuntos de productos que compiten en el mismo segmento de mercado de tal manera que existe una competencia feroz entre LPS. La definición de una MPL Competitiva se relaciona con el modelo de características independiente del proveedor *(SIFM, Supplier Independent Feature Model)* descrito en [46].

En una **MPL Composicional**, cada parte de la LPS corresponde a varias LPS, las cuales se combinan para formar un sistema de software integrado.

Tabla 1.2: Diferencias entre MPLs Competitivas y Composicionales

Término	MPL Competitiva	MPL Composicional
Suministro	Cada LPS pertenece a diferentes proveedores o empresas.	Cada parte de la LPS corresponde a varios LPS, construidos por diferentes proveedores o empresas.
Producto	Combinación de características que se proporcionan por un solo proveedor.	Combinación de características que se proporcionan por varias LPS diferentes.
Diferencias	En algunos casos, los productos de una MPL Competitiva son únicos.	El conjunto de productos de una MPL Composicional incluye el conjunto de productos de un MPL Competitiva.

La decisión de adoptar un enfoque tradicional, utilizar LPS o MPL depende de factores como [47]:

- Requisitos no funcionales.
- Lanzamiento de un producto a un segmento de mercado diferente y/o se requiere modificar.
- Adquirir una ventaja competitiva.

- Incorporar procesos de reutilización de componentes.
- Nuevas tecnologías de LPS.
- Mercados geográficos y logísticos.
- Compartir recursos.

1.1.3.2. Aplicación de MPL en diferentes dominios

Actualmente, existen diversos ejemplos que ilustran la idea original de utilizar MPLs, entre los cuales destacan principalmente en la Metalurgia, Siderurgia y los sistemas mecatrónicos.

1. **Metalurgia.** Actualmente, la mayoría de los metales empleados son aleaciones, es decir, mezcla de distintos metales que combinan sus propiedades para obtener materiales más ligeros y resistentes. En las aleaciones también intervienen la temperatura del horno de fundición y el procedimiento de enfriamiento. Por ejemplo, para que el Cobre sea versátil, se modifican sus características mediante la mezcla con otros metales en función del uso final deseado. De esta mezcla, es posible obtener más de 400 aleaciones, como Bronce, Latón, Cuproníquel, por mencionar algunos. Otra aleación identificada como Invar o FeNi36 se deriva del Níquel y Acero. Invar se utiliza en la fabricación de piezas de precisión como la relojería, los aparatos de física, las válvulas de motores y en instrumentos para medir longitud como los utilizados en topografía, esto debido a su bajo coeficiente de expansión térmica. Esta noción de combinar, integrar o componer diferentes enfoques motiva el enfoque denominado Invar (Integrated View on Variability) [48, 49, 50] para el desarrollo de software utilizando múltiples líneas de productos. Invar facilita el intercambio de modelos heterogéneos de variabilidad durante la configuración del producto independientemente de las técnicas, anotaciones y herramientas utilizadas en la organización.

2. **Siderurgia.** Otro ejemplo de implementación de MPL ocurre en la industria siderúrgica, específicamente en los mini molinos (*mini-mill*) ó acerías que conforman la cartera de productos de SIEMENS VAI [12]. Se denomina Siderurgia o Siderurgia integral a una

planta industrial dedicada al proceso completo de producir acero a partir del mineral de hierro, mientras que se denomina acería a una planta industrial dedicada exclusivamente a la producción y elaboración de acero a partir de otro acero o de hierro. Un *mini-mill* se construye a partir de varios sistemas interdependientes que en conjunto representan una Multilínea de Productos de Software. A diferencia de la siderúrgica integral, el *mini-mill* es una instalación que produce productos de acero utilizando chatarra como fuente de hierro. El mini-mill se integra por varias líneas de productos como el horno eléctrico, caster (máquina de colado continuo), tren de laminación y una herramienta de software para el Mantenimiento y Configuración del Sistema (MSS), utilizada por los clientes para personalizar la solución de software de un *mini-mill* durante la operación (Figura 1.10). Aunque los mini-mill están sujetos a los mismos requerimientos que las plantas de Siderurgia integral difieren en que son plantas flexibles con capacidad para actualizarse técnicamente, diversidad en los estilos de administración, relaciones laborales y mercados para el producto. El mini-mill y los diferentes subsistemas son personalizables en cuanto a la cantidad de hierro, tipo de horno, número de filamentos, tipo de tren de laminación o capacidad de laminación. A pesar de las dependencias, cada subsistema se trata como un único sistema y autónomo de la línea de productos.

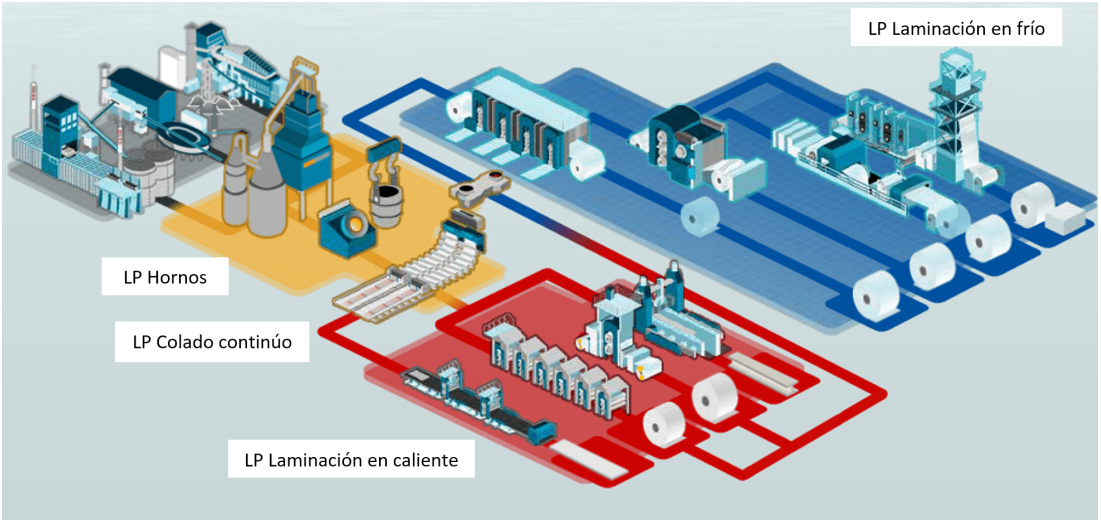


Figura 1.10: Ejemplo de un mini-mill

3. **Sistemas mecatrónicos.** La ingeniería mecatrónica combina varias disciplinas como ingeniería mecánica, electrónica, control automático y software para el diseño de productos y procesos. Los sistemas mecatrónicos tienen diversas aplicaciones: robótica, aeronáutica, industria automotriz, industria médica, domótica, entre otros. En el campo de la mecatrónica, los productos se describen mediante múltiples modelos pertenecientes a diferentes dominios de la ingeniería, como mecánico, eléctrico y electromecánico. En el dominio médico, se identifica el desarrollo de múltiples líneas de productos para los sistemas médicos de imágenes de Philips a través de líneas de productos jerárquicas [51] . Generalmente, hay varias líneas de productos disponibles porque los productos se desarrollan en diferentes partes del mundo y dentro de diferentes grupos de productos, como la resonancia magnética, los rayos X y la tomografía por ultrasonidos. Para manejar la complejidad en las líneas de productos para sistemas mecatrónicos, en [52] se propone un enfoque de MPL para distinguir entre software y hardware mediante el uso de diferentes modelos de características para cada uno.

La Tabla 1.3 muestra el análisis de los principales dominios de aplicación de las MPLs. La tabla destaca las principales similitudes y diferencias en las MPLs aplicadas a la industria para la fabricación de productos físicos y productos de software.

Tabla 1.3: Dominios de aplicación de las MPLs

Dominio / Elemento	**Metalurgia**	**Siderurgia**	**Sistemas mecatrónicos**	**Ingeniería de Software**
Activos	Metales	Chatarra de acero	Hardware y Software	Artefactos de Software (requisitos, modelo de características, arquitectura, bibliotecas, pruebas, etc.)
Reutilización	Procesos, máquinas y herramientas			Procesos y herramientas de software

Continúa en la página siguiente

Tabla 1.3:Dominios de aplicación de las MPLs (continuación)

Dominio	**Metalurgia**	**Siderurgia**	**Sistemas mecatrónicos**	**Ingeniería de Software**
Proveedores	Varios			
Variabilidad	Si			
Capacidad de producción	Limitada por la infraestructura			Ilimitada, pero depende de la configuración del producto
Balanceo de la línea	Si			
Aplicación	Aleaciones	Mini-mill	Philips Medical Imaging Systems, Aselsan REHIS	SECO, SoS, ERP, Cadena de Suministro de Software

Es importante destacar que en una MPL aplicada a la industria manufacturera se considera la capacidad de producción, porque la cantidad de productos o servicios que es posible obtener por un período de tiempo depende de la demanda e infraestructura de la empresa. En caso contrario, la capacidad de una MPL en el contexto del software es ilimitada porque es posible generar *N* productos de software.

Los cuatro dominios de aplicación de las MPL coinciden en que es posible involucrar a diferentes proveedores, obtener diferentes versiones de productos (variabilidad) y reutilizar procesos y herramientas.

Otro aspecto a resaltar en las MPLs en el contexto del software es que al igual que las LPS no solo se ocupan de la generación de productos de software sino que también generan otros artefactos como requisitos, código, arquitectura, pruebas, documentación, por mencionar algunos.

MPL es un enfoque multidominio que surge para desarrollar sistemas grandes y complejos a través de varias líneas de productos independientes, es decir que son desarrolladas por varias organizaciones con diferentes enfoques y tecnologías para diversas áreas geográficas y en cualquier contexto. Este enfoque se investiga para el desarrollo de Sistemas de Sistemas (SoS) y

Ecosistemas de Software (SECO) que resultan de la integración de varios sistemas operacionalmente independientes. La viabilidad del enfoque y su implementación es útil para sistemas ERP o cadenas de suministro de software.

1.1.3.3. Requisitos de las MPL

Los requisitos fundamentales para cualquier enfoque que apoye a las Multilíneas de Productos de Software son [53]:

1. **Heterogeneidad y distribución:** se refiere a que es necesario tratar a las LPS como proyectos independientes e integrarse según sea necesario en el contexto de producto derivado. Esto requiere que cada LPS tenga su propia descripción de variabilidad, sus propios artefactos y su propio enfoque para la creación de instancias.

2. **Manejo de la composición con mínima complejidad:** se refiere a que el desarrollador del producto maneja fácilmente los artefactos instanciados de las LPS. La composición en los artefactos y en el modelo de variabilidad requiere ser de forma sincronizada y de fácil acceso para el desarrollador que trabaja a nivel del producto.

3. **Apoyo a la funcionalidad específica del producto con mínima complejidad**: se refiere a que el desarrollador del producto maneje productos específicos independientemente de los asuntos sobre las LPS reutilizadas.

Las capacidades necesarias identificadas por Holl [10] que se requieren para apoyar a las MPLs son:

1. Soporte para la derivación de productos concurrentes y distribuidos.

2. Compartir y distribuir modelos de variabilidad de las Líneas de Productos.

3. Soporte de implementación para modelos y herramientas de Líneas de Productos.

4. Definición y gestión de usuarios y roles involucrados en las MPLs.

5. Soporte para la estructuración de modelos de Líneas de Productos.

6. Métricas y visualización.

7. Guía del usuario en la derivación.

8. Definir y gestionar diferentes tipos de dependencias entre Líneas de Productos.

9. Soporte para la definición de interfaces explícitas entre Líneas de Productos.

10. Control de consistencia en Multilíneas de Productos.

11. Modelado de lenguajes para sistemas a gran escala.

1.1.3.4. Ciclo de vida de una MPL

La Figura 1.11 presenta el ciclo de vida para las MPL propuesto por Holl [20] basado en un escenario que abarca las tareas de modeladores, arquitectos de sistemas y configuradores.

1. **Modelado:** los modeladores crean modelos de variabilidad de LPS para los sistemas involucrados en una MPL y comparten un repositorio de modelos.

2. **Definición:** el arquitecto utiliza los modelos individuales de LPS para definir y componer una arquitectura de referencia para MPL que representa el SoS. Por otra parte, el arquitecto define dependencias de configuración entre LPS relacionados con la arquitectura.

3. **Aprobación:** el arquitecto del sistema verifica que la arquitectura de referencia es consistente con el SoS.

4. **Instanciación:** el arquitecto crea una configuración MPL basado en la arquitectura de referencia mediante la selección e instanciación de los modelos de las LP necesarios para cumplir con los requisitos de alto nivel de un cliente.

5. **Asignación:** el arquitecto despliega la configuración de la MPL a los configuradores a traves de un repositorio de modelos compartidos.

6. **Configuración:** los configuradores realizan la configuración utilizando los modelos instanciados de las LPS asignados por el arquitecto del sistema.

7. **Distribución:** los configuradores ponen a disposición de sus colabores los modelos de las LPS a través de un repositorio.

8. **Generación:** los configuradores desarrollan un producto basado en la configuración actual.

9. **Evolución:** los modeladores adaptan los modelos de variabilidad de las LPS para cumplir con los nuevos requisitos.

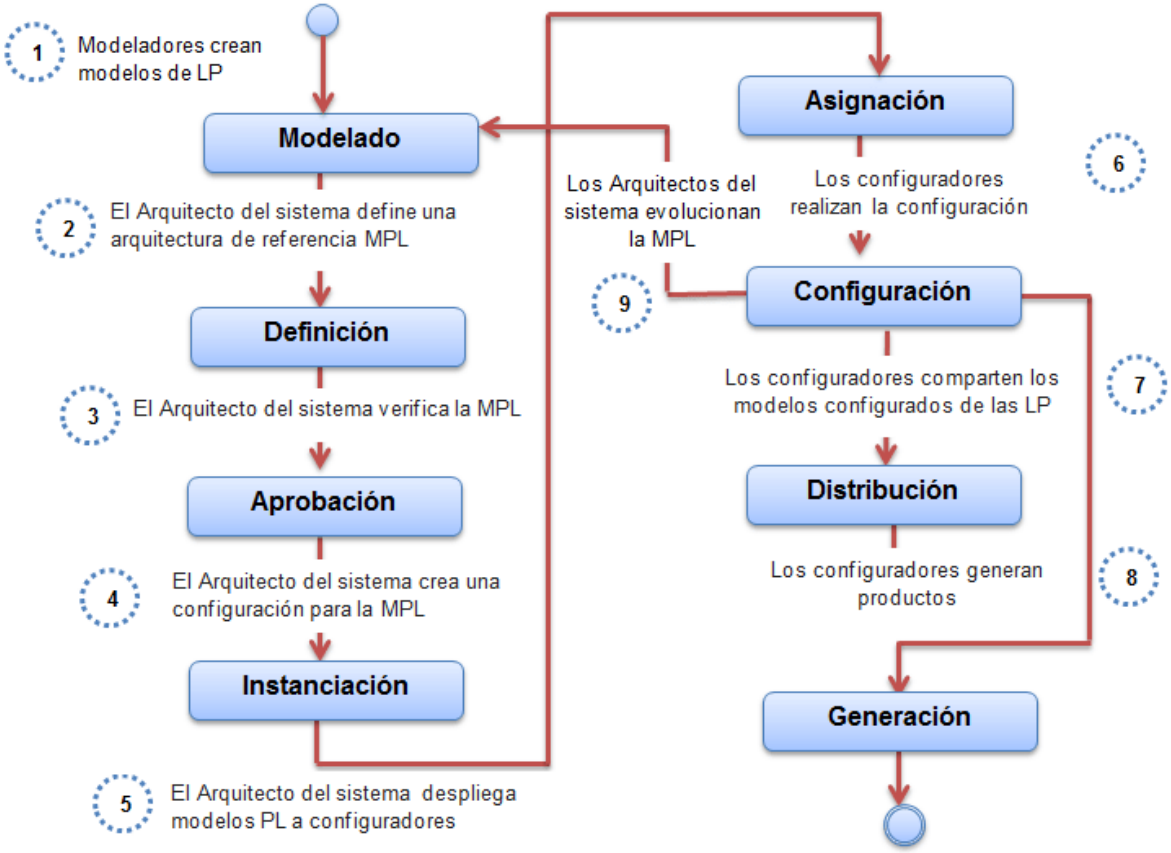


Figura 1.11: Ciclo de Vida de una MPL

1.1.3.5. Desafíos de las MPL

La aplicación de MPL en el contexto del software se encuentra en desarrollo y requiere atender una serie de desafíos que se presentan cuando las LPS evolucionan o requieren man-

tenimiento para satisfacer las necesidades cambiantes del mercado ya sea por funcionalidad, enfoque o tecnología minimizando costos (tiempo y dinero) y esfuerzo de desarrolllo utilizando un esquema de reutilización similar a las LPS. Entre los principales desafíos a enfrentar al utilizar MPLs (Figura 1.12), identificados en la literatura destacan:

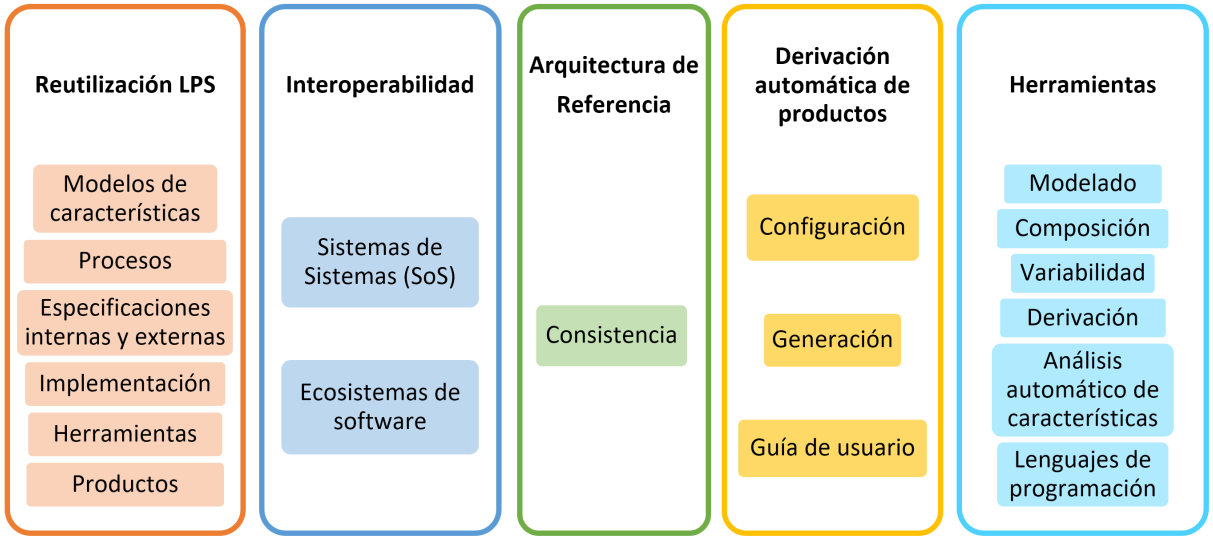


Figura 1.12: Clasificación de los desafíos de las MPL

Desafío 1. Se requiere manipular configuraciones complejas, manejar una adecuada separación de asuntos y guiar al usuario [54].

Desafío 2. Se requieren técnicas, métodos o enfoques que faciliten y maximicen la reutilización de los artefactos de software [2] de las LPS para generar una familia productos válidos en una MPL. Esto implica la gestión de funcionalidad dispersa (asuntos de corte) en diversos dominios de las LPS [55]. La composición de LPS [56] permite estructurar y descomponer LPS a gran escala y facilita la reutilización de LPS dentro de otras LPS.

Desafío 3. Adaptación y extensión de herramientas LPS para el modelado de características, composición de LPS, configuración y derivación de productos de una MPL [14,

[2] Artefacto de software: producto tangible resultante del proceso de desarrollo de software como, por ejemplo: casos de uso, diagrama de clases, código fuente, documentación, arquitectura, entre otros.

57, 19]. Además, se requieren herramientas que faciliten el análisis automático de modelos de características dependientes [58].

Desafío 4. Comprensión del dominio y reducir la complejidad del análisis de LPS [59].

Desafío 5. Facilitar la fusión y reutilización de modelos de características provistos por diferentes compañías o proveedores independientemente de las técnicas de modelado, notaciones y herramientas utilizadas [60, 8, 33, 49]. Es necesario, la representación de un modelo de caraterísticas universal para gestionar y combinar la variabilidad de las LPS que integran la MPL [61].

Desafío 6. Se requiere promover la interoperabilidad[3] entre diversas LPS y sus productos [1, 62]. Una interoperabilidad adecuada es útil para para promover la cooperación entre sistemas independientes con el fin de proporcionar funciones más complejas (SoS o SECO), las cuales no son proporcionadas por algún sistema que funcione por separado.

Desafío 7. Definir una arquitectura de referencia [44] o arquitectura MPL consistente que represente las partes comunes y diferentes de las LPS para el desarrollo incremental de productos [63]. Se requiere que la arquitectura sea capaz de satisfacer nuevos requisitos, es decir que sea posible la implementación de nuevas características o su modificación en la MPL (evolución) [38, 22].

Desafío 8. Comprobar la consistencia de la arquitectura de la MPL [45]. La consistencia de la arquitectura de la aplicación con la arquitectura de la LPS es importante para asegurar que las reglas de negocio y las restricciones que se definen para toda la familia de productos no se violen. Normalmente, la comprobación de la consistencia con la arquitectura de la línea de productos es un proceso manual y tedioso. La consistencia de la arquitectura implica que los elementos de diseño de la arquitectura se mapean a los elementos de implementación.

[3] La IEEE define a la interoperabilidad como la habilidad de dos o más sistemas o componentes para intercambiar información y utilizar la información intercambiada.

Desafío 9. El análisis del impacto del cambio (CIA) en el contexto MPL no está bien estudiado aún [64]. Por ejemplo, se requiere calcular el impacto de un cambio de una característica en las configuraciones existentes de una MPL [65]. Sin embargo, la CIA es útil para la identificación y análisis de cambios, así como para la estimación de la necesidad de modificación.

Desafío 10. Automatizar la configuración (colaborativa) y derivación de productos en una MPL [66, 67, 56, 68, 14, 50]. La configuración en una MPL es un proceso de colaboración entre diferentes personas y equipos con diversos conocimientos en relación con los dominios, notaciones y lenguajes utilizados. El problema surge porque se requiere asignar las decisiones adecuadas a las partes apropiadas en el momento y orden adecuado basándose en el dominio de los conocimientos de las personas o equipo de desarrollo. También, es necesario considerar las dependencias entre opciones de configuración de diferentes LPS que constituyen una MPL [19].

Desafío 11. Asegurar la transferencia de conocimiento a los múltiples usuarios involucrados en la derivación de productos en una MPL. Esto se refiere a que los usuarios que configuran el sistema estén conscientes de las decisiones tomadas [12].

1.1.4. Inteligencia Artificial

La Inteligencia Artificial se centra en el diseño e implementación de programas autónomos inteligentes que tratan de simular las acciones racionales de los seres humanos. A diferencia, la Ingeniería de Software es una disciplina que integra métodos, herramientas y procedimientos para el desarrollo de software de computadora. Sin embargo, los algoritmos de Inteligencia Artificial se adaptan bien a problemas complejos de la Ingeniería de Software, ya que están diseñados para afrontar uno de los desafíos más importantes: la replicación del comportamiento inteligente [69].

Los problemas de la Ingeniería de Software se caracterizan porque [70]:

- Es necesario equilibrar las restricciones competitivas.

- Ocasionalmente, existe la necesidad de hacer frente a las inconsistencias.
- A menudo hay muchas soluciones potenciales.
- Normalmente no hay una respuesta perfecta pero es posible reconocer a las buenas soluciones.
- A veces no hay reglas precisas para calcular la mejor solución.

Actualmente, la comunidad de Ingeniería de Software utiliza y adapta algoritmos, métodos y técnicas prácticas que han surgido de la comunidad de Inteligencia Artificial. Estos algoritmos y técnicas de Inteligencia Artificial tienen aplicaciones importantes y efectivas que impactan en casi todas las áreas de la Ingeniería de Software. En particular, la comunidad de Ingeniería de Software utiliza tres áreas de la Inteligencia Artificial:

1. Ingeniería de Software Basada en Búsqueda (SBSE).
2. Métodos difusos y probabilísticos para razonar en presencia de incertidumbre.
3. Clasificación, aprendizaje y predicción.

1.1.4.1. Ingeniería de Software Basada en Búsqueda

La **Ingeniería de Software Basada en Búsqueda (*SBSE, Search-based software engineering*)** es una disciplina emergente en la cual los problemas de Ingeniería de Software se reformulan y modelan como problemas de optimización, y posteriormente, se resuelven utilizando conceptos, técnicas, algoritmos y estrategias de búsqueda. Estos problemas se caracterizan por un gran espacio de búsqueda, en el que múltiples objetivos y restricciones están involucrados y es posible que estén en conflicto. El objetivo de la búsqueda es identificar, entre todas las posibles soluciones, una que sea lo suficientemente buena según las métricas apropiadas [70, 71, 69].

SBSE se basa principalmente en técnicas de Computación Evolutiva como los algoritmos genéticos, ascensión de colinas, recocido simulado, colonia de hormigas, enjambre de partículas, entre otros. Estas técnicas son genéricas y robustas, las cuales han demostrado escalar a espacios

de búsqueda grandes y se aplican generalmente en la ingeniería de requisitos, diseño, desarrollo, pruebas y mantenimiento de software.

Harman y Clark [71] identifican cuatro propiedades importantes para que el enfoque de Ingeniería de Software Basada en la Búsqueda sea exitoso:

1. Gran espacio de búsqueda.
2. Baja complejidad computacional.
3. Continuidad aproximada.
4. Ausencia de soluciones óptimas conocidas.

Generalmente, estas cuatro características de problemas son frecuentes en la Ingeniería de Software, especialmente en aquellos que implican un gran espacio de búsqueda como la cantidad de diseños posibles, casos de prueba o configuraciones del sistema. Además, en diversas situaciones, no se conoce una solución óptima para el problema. Las propiedades de "baja complejidad computacional"y "continuidad aproximada"es posible que no estén presentes en todos los casos. Sin embargo, incluso en los casos en que están ausentes, es posible transformar el problema en uno que sea más susceptible a la Ingeniería de Software Basada en la Búsqueda.

1.1.4.2. Reformular la SPLE como un problema de búsqueda

Es natural pensar que varios de los problemas en la Ingeniería de Líneas de Productos de Software son problemas de optimización. Esto debido a que la variabilidad de los diferentes productos expresada por sus modelos de características, crea un gran espacio de búsqueda en el que es posible buscar opciones de productos óptimas (o casi óptimas).

En la literatura, existen diversas publicaciones que exploran la aplicación de técnicas de SBSE en diversas actividades del ciclo de vida de las LPS como la selección de características óptimas, la generación de pruebas, la arquitectura, entre otros. Estas técnicas y aplicaciones resultan interesantes para enfrentar algunos desafíos que es posible surjan al implementar una MPL, como la reutilización de LPS, la interoperabilidad, la derivación de productos, la arquitectura de referencia, entre otras.

Para reformular los diversos asuntos de la Ingeniería de Líneas de Productos de Software como un problema de búsqueda, es necesario definir:

- La representación del problema susceptible de manipulación simbólica.
- La función de aptitud (*fitness function*) para cuantificar la optimalidad de una solución para que una solución en particular se compare con todas las demás soluciones.
- Un conjunto de operadores para manipular las soluciones.

1.1.4.3. Identificación de técnicas de SBSE en las LPS

Durante la última década, el alcance de la Ingeniería de Software Basada en Búsqueda se ha extendido para cubrir diferentes dominios del ciclo de vida de las LPS, tales como: selección de características, generación de pruebas, arquitectura, mantenimiento, entre otros. La Tabla 1.4 muestra los problemas de optimización identificados en las LPS y las técnicas de SBSE utilizadas para resolverlos.

Tabla 1.4: Técnicas SBSE

Dominio del problema	SA	GA	ACO	Pairwise	BA	MOO	FL	RS
Selección de características / Configuración de productos	Si [72]	Si [73, 74, 75, 76, 77]	Si [78]	-	-	Si [74]		[79]
Arquitectura	-	-	-	-	-	Si [80, 81]		
Mantenimiento y evolución Ubicación de características	-	Si [82]	-	-	-		-	
Pruebas	-	Si [83, 84]	-	Si [85]	Si [86]	Si [83]		
Alcance de la plataforma de productos	Si [87]	-	-	-	-			
Gestión de prioridades	-	-	-	-	-	-	Si [88]	

A continuación se describen algunas técnicas de SBSE y cómo se utilizan para resolver diferentes problemas o asuntos identificados en el dominio de las LPS:

El **recocido simulado (*SA, Simulated Annealing*)** es un método de búsqueda aplicado a problemas de optimización combinatoria. Se denomina así porque está inspirado en el proceso físico de recocido de sólidos, el cual utiliza un procedimiento que va disminuyendo la temperatura, con lo cual se modifica la estructura del material. El enfriamiento debe hacerse de manera lenta para obtener configuraciones moleculares resistentes. Cada etapa del enfriamiento tiene asociada una energía y una configuración del material determinadas. En [87], esta técnica se utiliza para optimizar el alcance de una plataforma de producto de software.

Los **algoritmos genéticos (*GA, Genetic algorithm*)** son técnicas utilizadas para resolver problemas de búsqueda y optimización que simulan el proceso de evolución natural. Los algoritmos genéticos se basan en la teoría de Darwin, en la cual los individuos más aptos sobreviven, mientras que los menos adaptados tienden a desaparecer. El procedimiento de búsqueda (Figura 1.13) tiene el propósito de mantener una población de soluciones potenciales (cromosomas) mientras se realiza una investigación paralela para soluciones con una función de aptitud alta. Los algoritmos genéticos tienen varias variaciones, específicamente para los operadores genéticos (cruce, mutación), la selección y cómo se reemplaza a los individuos para formar la nueva población. Existen tres pasos principales en un algoritmo genético: crossover, mutación y selección. En la literatura, se proponen muchas variantes para el operador de crossover, pero el principio común es combinar dos cromosomas para generar cromosomas de próxima generación, mediante un intercambio de genes simple o no, con pequeñas variaciones. La mutación cambia aleatoriamente los valores del gen para generar una nueva combinación de genes para la próxima generación. Matemáticamente, el interés principal de la mutación consiste en saltar de soluciones óptimas locales. La selección es el último paso donde se copian las mejores soluciones cromosomicas en la próxima generación.

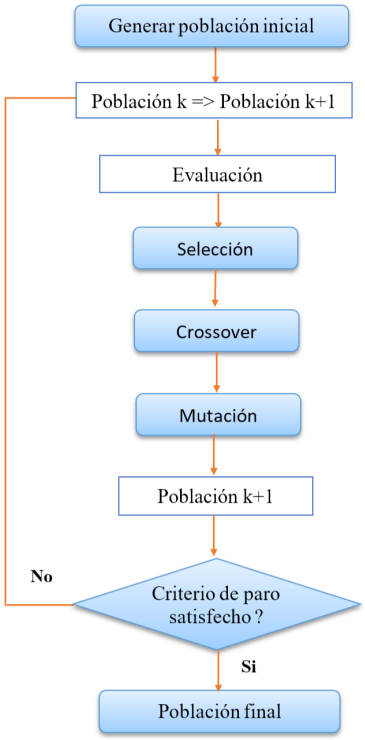


Figura 1.13: Algoritmo genético

En [82], los autores presentan un enfoque llamado *GA-FL (Genetic Algorithm to Feature Location)* para la ubicación de características en LPS basadas en modelos. La ubicación de características (*FL*) se conoce como el proceso de encontrar el conjunto de artefactos de software que realizan una característica particular.

En [84] abordan la minimización de pruebas para las LPS mediante la identificación y eliminación de casos de prueba redundantes de las suites de prueba con el fin de reducir la cantidad total de casos de prueba a ejecutar, mejorando así la eficiencia de la prueba. Para lograr esto, formulan el problema de minimización como un problema de búsqueda y definen una función de aptitud considerando varios objetivos de optimización. Para evaluar el desempeño de la función de aptitud utilizan algoritmos genéticos.

La **optimización por colonia de hormigas** ***(ACO, Ant Colony Optimization)*** es una técnica de optimización aproximada para resolver problemas computacionales que buscan los mejores caminos o rutas en grafo. La fuente inspiradora de los algoritmos ACO son las colonias de hormigas. Wang y Pang presentaron un algoritmo polinomial para la selección de características con restricciones en las LPS usando ACO [78]. Los autores proponen un método para transformar un modelo de características de y / o árbol a gráficos dirigidos de varias etapas. Luego, el problema de selección de características se convierte para encontrar una ruta óptima desde el origen hasta el destino.

La **prueba por pares (*Pairwise Testing*)** de LPS tiene como objetivo seleccionar un conjunto de productos (conjunto de pruebas), de modo que su cobertura combinada contenga todas las posibles combinaciones de características por pares de la LPS.

El **algoritmo de murciélago (*BA, Bat algorithm*)** es uno de los algoritmos heurísticos propuestos recientemente que imitan el comportamiento de ecolocación de los murciélagos para realizar una optimización global. BA es la combinación de enjambre y un algoritmo de ruta. En [86], se presenta un enfoque llamado SPLBA para la reducción de casos de pruebas para LPS utilizando un algoritmo inspirado en los murciélagos.

La **optimización multi-objetivo** ***(MOO, Multi-objective optimization)*** también denominada optimización multi-criterio considera problemas de optimización que requieren optimizar simultáneamente más de una función objetivo. Un diseño de arquitectura de LPS es un problema de optimización que se ha resuelto de manera efectiva utilizando algoritmos multi-objetivos para encontrar soluciones óptimas que satisfagan los objetivos definidos. Por ejemplo, MOA4PLA (Multi-Objective Optimisation Approach for PLA design) es un enfoque basado en la búsqueda para respaldar las arquitecturas de LPS [89]. MOA4PLA utiliza un PLA modelada en un diagrama de clase UML y lo optimiza. Se genera un conjunto de soluciones con la mejor compensación entre objetivos y el arquitecto selecciona la arquitectura a utilizar. En [83], se presenta un enfoque para manejar múltiples objetivos en conflicto en la generación de pruebas para LPS. Este

enfoque combina algoritmos genéticos y técnicas de resolución de restricciones para tratar los siguientes objetivos: 1) maximizar la cobertura por pares, 2) minimizar la cantidad de productos seleccionados y 3) minimizar el costo total del conjunto de pruebas.

La **Lógica Difusa (*Fuzzy Logic*)** proporciona un mecanismo de inferencia que permite simular los procedimientos del razonamiento humano en sistemas basados en el conocimiento. La teoría de la lógica difusa proporciona un marco matemático que permite modelar la incertidumbre de los procesos cognitivos humanos proporcionando herramientas formales para su tratamiento. Básicamente, cualquier problema se resuelve a partir de un conjunto de variables de entrada (espacio de entrada) y se obtiene un valor adecuado de variables de salida (espacio de salida). La lógica difusa permite establecer este mapeo de una forma adecuada, atendiendo a criterios de significado (y no de precisión). Actualmente, se utiliza en un amplio sentido, agrupando la teoría de conjunto difusos, reglas si-entonces, aritmética difusa, cuantificadores, entre otros. En [88] los autores utilizan la lógica difusa para asignar pesos (prioridades) a las características en un modelo de características. La asignación de prioridades en un modelo de características ayuda a los desarrolladores a seleccionar las características de alta prioridad sobre las más bajas. Definen los tipos de la función de pertenencia, número y los tipos de distribución de los conjuntos difusos.

Los **Sistemas de Recomendación (RS)** aprenden de las preferencias de los usuarios y predicen futuros elementos de interés para ellos. Su objetivo es aliviar el problema de la sobrecarga de información que se produce en la configuración de la LPS, donde el número de características y configuraciones posibles es demasiado alto para que un solo usuario lo manipule. En [79] proponen un Sistema de Recomendación de características colaborativo, el cual se basa en configuraciones de usuarios anteriores para generar recomendaciones personalizadas para un usuario actual. Esto es debido a que la configuración manual de un producto es un proceso masivo y difícil. Para facilitar este proceso, adaptan tres algoritmos de recomendación personalizada con el escenario de la configuración de la LPS: vecino más cercano, similitud media y factorización de matrices. Estos algoritmos utilizan configuraciones anteriores para estimar y predecir la re-

levancia de las características con el fin de guiar al usuario a través del proceso de selección de características.

1.2. Planteamiento del problema

Las Líneas de Productos de Software (LPS) cada vez son más utilizadas[4] con el objetivo de reducir costos de desarrollo y aumentar la productividad a partir de la reutilización de componentes y artefactos en varios productos. La mayoría de enfoques para la Ingeniería de Línea de Productos de Software se basan en el desarrollo de una sola plataforma enfocada a un único segmento de mercado. Sin embargo, como los requisitos y necesidades del cliente cambian frecuentemente, es necesario agregar y configurar nuevos productos en la LPS. Los especialistas en el desarrollo de LPS reconocen que no es posible extender o adaptar la plataforma indefinidamente. Una alternativa es desarrollar Multilíneas de Productos de Software (MPL) que permiten construir aplicaciones con funcionalidades complejas (Sistemas de Sistemas o Ecosistemas de Software) capaces de satisfacer las necesidades de diferentes segmentos de mercado a partir de la reutilización, combinación y gestión de la variabilidad de los insumos provistos por diversas LPS. Asimismo, las MPL reducen los riesgos de las compañías de desarrollo al brindar soporte para la introducción de nuevas tecnologías o la modificación gradual de la infraestructura de la LPS con menos riesgos ya que seguirán funcionando normal e independientemente de los resultados obtenidos con la MPL.

A pesar de la experiencia que se tiene en la industria (*mini-mill*), la implementación de MPL en el contexto del software se encuentra en desarrollo y las investigaciones actuales se encuentran fragmentadas debido a la diversidad de conceptos relacionados y reportan una serie de desafíos (sección 1.1.3.5) que requieren atenderse para utilizar este enfoque. En este sentido, para que las compañías sean capaces de satisfacer nuevos requisitos o modificar sus productos

[4] Se considera que las LPS son empleadas por empresas que cuentan con años de experiencia en el desarrollo de software y que cada vez más se especializan enfocándose a segmentos de mercado.

sin comprometer su infraestructura y de acuerdo al contexto de la presente tesis, es de vital importancia atender los siguientes desafíos:

- Analizar el potencial y la capacidad de combinación y reutilización de los insumos de las LPS (Desafío 2). Para ello, se requiere conocer las características de las LPS que permiten obtener configuraciones válidas (selección/re-configuración de características) y así derivar automáticamente nuevos productos de software en una MPL (Desafío 10).

- Debido a que la configuración en una MPL es un proceso de colaboración entre diferentes personas y equipos con diversos conocimientos en relación con los dominios, notaciones y lenguajes utilizados, se requiere promover la interoperabilidad entre las diversas LPS y sus productos (Desafío 6).

- Para promover la interoperabilidad de las LPS, es necesario definir una arquitectura de referencia consistente (Desafío 7) que represente las partes comunes y diferentes de las LPS para el desarrollo incremental de productos.

1.3. Hipótesis

H_i: El análisis de los modelos de características utilizando estrategias de Ingeniería de Software Basada en Búsqueda apoyará la derivación de nuevos productos de software válidos [5] para el desarrollo de Multilíneas de Productos de Software.

1.4. Objetivos de la investigación

A continuación, se presentan tanto el objetivo general como los objetivos específicos.

[5] Un producto es una configuración válida para una LPS si se configura utilizando las características de la LPS y además es una instancia de su modelo de características (ver Sección 1.1.2.6)

1.4.1. Objetivo general

Desarrollar un modelo matemático [6] para apoyar la derivación de productos en una Multilínea de Productos de Software utilizando técnicas de Ingeniería de Software Basada en Búsqueda.

1.4.2. Objetivos específicos

- Identificar los artefactos que intervienen en el desarrollo de una MPL para aplicarlos en el contexto del software.
- Identificar las herramientas utilizadas en las LPS para apoyar el desarrollo de una MPL.
- Identificar las técnicas de Ingeniería de Software Basada en Búsqueda para resolver problemas en el ciclo de vida de las LPS.
- Definir las estrategias de composición para la implementación de una MPL en el contexto del software.
- Analizar los enfoques disponibles en la literatura para representar las Arquitecturas de Referencia en las MPL.
- Definir un caso de estudio para el diseño e implementación del modelo matemático y la MPL en el área de Inmótica como prueba de concepto.
- Definir una Arquitectura de Referencia para representar los activos comunes y variables de las LPS que integran la MPL.
- Diseñar un modelo matemático para optimizar el proceso de configuración de productos en una MPL.
- Desarrollar escenarios óptimos de configuración (selección de características) de productos en una MPL a través del modelo matemático.

[6] Según la RAE un modelo es un esquema teórico, generalmente en forma matemática, de un sistema o de una realidad compleja, que se elabora para facilitar su comprensión y el estudio de su comportamiento.

- Desarrollar un generador (configurador) de productos de la MPL Inmótica para generar aplicaciones utilizando los escenarios óptimos de configuración de configuración.

1.5. Aportaciones de la investigación

Las aportaciones del proyecto de investigación son:

- Modelo matemático para la selección de características de las LPS que facilite la toma de decisiones durante la fase de configuración (combinación) de artefactos de software en una MPL.
- Arquitectura de Referencia de la MPL que represente las partes comunes y variables de las LPS para el desarrollo incremental de productos.
- Una Multilínea de Productos de Software (MPL) como una forma para obtener y/o complementar productos de software a partir de la combinación y reutilización de los insumos de las LPS disponibles.

1.6. Justificación

Una Multilínea de Productos de Software (MPL) tiene como objetivo la derivación de nuevos productos de software a partir de un conjunto de características reutilizables provistas por diferentes LPS heterogéneas sin modificar ni poner en riesgo el funcionamiento de las mismas.

Para la derivación de productos en una LPS, un ingeniero de aplicaciones recibe un modelo de características y los requisitos de la aplicación e intenta seleccionar un subconjunto de características que genere el producto de software requerido y personalizado a las necesidades y preferencias de los stakeholders. Esta cuestión se conoce como el problema de selección óptima de características (optimal feature selection) [73, 78, 90, 91, 92] o problema de la configuración [93] y se convierte en un problema de optimización que plantea desafíos para el razonamiento y la configuración de las características.

En la literatura de LPS, el proceso de derivación de productos de software se modela como la optimización de la selección de características con restricciones de recursos, el cual es un problema no determinista duro polinomial (NP-hard) [94]. La optimización aplicada a las LPS es un medio para encontrar la mejor variante de productos (selección de características) conforme a propiedades funcionales y no funcionales [95, 96] específicas durante la Ingeniería de Aplicaciones. Sin embargo, los diseñadores y arquitectos toman sus decisiones basándose en múltiples propiedades (costo, viabilidad técnica, confiabilidad, entre otros) al mismo tiempo, por lo que la configuración de LPS se refiere a un problema de optimización multi-objetivo [97, 98].

Durante la fase de configuración, el ingeniero de productos requiere de técnicas, métodos y herramientas que faciliten el proceso de configuración del producto a partir de los modelos de características. Si bien existen diversas herramientas especializadas que proporcionan soporte a la configuración de LPS como por ejemplo: FeatureIDE, SPLOT o FaMa, sólo garantizan la configuración parcial de un producto y el ajuste a las restricciones de la LPS es de forma manual. Algunas veces, esto ocasiona retrasos debido a la exploración de opciones de los usuarios en cada paso del proceso de configuración. La configuración de productos en una MPL es de vital importancia debido a que los modelos de características son más grandes que para las LPS y las dependencias entre varias LPS introducen mayor complejidad. No obstante, el proceso de configuración de productos en una MPL es un problema crítico y la literatura actual carece de enfoques que apoyen este proceso de forma completa.

La configuración de una MPL se refiere a un problema de optimización combinatoria no lineal y multi-objetivo, porque intervienen cierto número de variables (características) y restricciones donde cada variable tiene n diferentes valores y es posible que pertenezca a diferentes modelos de características y cuyo número de combinaciones crece exponencialmente, lo que da lugar a múltiples soluciones óptimas para una instancia (producto). Para ello, es de vital importancia el diseño de un modelo matemático que permita representar la configuración de una MPL y facilite la derivación de productos en una MPL. Esto debido a que visualizar las diferentes configuraciones válidas de productos es una tarea compleja y tediosa al tener disponibles una gran cantidad de características y restricciones las cuales además originan configuraciones inválidas, inapropiadas o ineficientes.

El empleo de técnicas de Ingeniería de Software Basada en Búsqueda como los algoritmos genéticos para optimizar el desarrollo de MPLs resultan una alternativa viable a los desarrolladores de software para tomar decisiones de diseño eficientes en relación a cómo combinar insumos de la MPL (LPS y artefactos) para desarrollar productos de software.

Capítulo 2

Estado del arte

Este capítulo presenta la revisión de la literatura y el análisis de los trabajos relacionados directamente con el proyecto de investigación con el fin de obtener información relevante y conocer el estado actual de las Multilíneas de Productos de Software. La sección 2.1 describe el reto de la derivación de productos en las LPS. La sección 2.2 presenta los trabajos que atienden el desafío 2 referente la reutilización de LPS. La sección 3.6 comprende artículos que atienden los desafíos 6 y 7 de la (sección 1.1.3.5) y que se refieren a cómo promover la interoperabilidad de las LPS que integran una MPL mediante arquitecturas de referencia. La sección 2.4 comprende los trabajos que atienden la configuración de productos en una MPL (desafío 10). Finalmente, la sección 2.5 describe el panorama general de las Multilíneas de Productos de Software.

2.1. Derivación de productos en las LPS

El proceso de derivación del producto es un "problema no determinístico duro polinomial"*(NP-hard, Nondeterministic polynomial-time hard)* y se modela como la optimización de la selección de características con restricciones de recursos [94].

Durante la derivación de productos, un ingeniero de aplicaciones recibe un modelo de características y los requisitos de la aplicación e intenta seleccionar un subconjunto de las características que optimizan los requisitos. Resolver el problema de configuración es un reto complejo debido a las siguientes razones [93]:

- Existen varios tipos de relaciones de variabilidad y restricciones de integridad entre las características.
- El número de posibles configuraciones es muy alto incluso en modelos de características pequeños. La complejidad de la configuración incrementa cuando se trata de modelos de características industriales que incluyen cientos de características.
- Existe la posibilidad de que las características tengan un impacto positivo o negativo en los diferentes asuntos comerciales de un producto, y por lo tanto expongan diferentes atributos de calidad [99]. Por ejemplo, es posible que una característica tenga un impacto negativo en la calidad, pero un impacto positivo en la satisfacción del cliente o tener alto desempeño pero baja calidad.
- Además de los requisitos funcionales, los stakeholders tienen varias limitaciones y preferencias sobre las propiedades no funcionales durante la derivación del producto, por lo que es necesario priorizar las características.
- Múltiples requisitos no funcionales competitivos y conflictivos. Es decir, existen relaciones de interdependencia entre propiedades no funcionales debido a que es posible que un aumento o disminución en el valor de una propiedad no funcional conduzca a un aumento o disminución en el valor de otra propiedad no funcional. Por ejemplo, es posible que el aumento del valor de la seguridad disminuya el valor del rendimiento para las partes interesadas

2.1.1. Selección de características

La selección de características consiste en elegir la "mejor"configuración de características que genere el producto de software requerido y personalizado a las necesidades y preferencias de los stakeholders. Esta cuestión conocida como el problema de selección óptima de características (optimal feature selection) [73, 78, 90, 91, 92] o problema de la configuración [93] se convierte en un problema de optimización que plantea desafíos para el razonamiento y la configuración de los modelos de características. La optimización aplicada a las LPS es un medio

para encontrar la mejor variante de productos (selección de características) conforme a propiedades funcionales y no funcionales [95, 96] específicas durante la Ingeniería de Aplicaciones. Sin embargo, al considerar múltiples propiedades al mismo tiempo se trata de un problema de optimización multi-objetivo [97, 98].

En la literatura se han propuesto diferentes algoritmos evolutivos (MEOAs, Multiobjective Evolutionary Optimization Algorithms) como IBEA (Indicator Based Evolutionary Algorithm), NSGAII (Non-dominated Sorting Genetic Algorithm-II), MOCell (Multi-Objective Cellular Genetic Algorithm), SPEA2 (Strength Pareto Evolutionary Algorithm 2) para resolver problemas de optimización multi-objetivos en la Ingeniería de Líneas de Productos de Software (SPLE) [75, 100] y satisfacer los requerimientos del cliente al encontrar una selección óptima de características. A continuación, se presentan los trabajos identificados en la literatura para apoyar y optimizar la selección de características y configuración de productos en una LPS.

La selección de características para el problema de derivación del producto se describió por primera vez en [94]. White y colaboradores proponen un método denominado Filtered Cartesian Flattening (FCF) para seleccionar conjuntos de características óptimas en un modelo de características de una LPS con restricciones de recursos. Asimismo, Guo [73] considera la restricción de recursos e introduce un enfoque denominado GAFES basado en algoritmos genéticos.

Guo [73] introduce un enfoque denominado GAFES basado en algoritmos genéticos para optimizar la selección de características en un modelo de características de una LPS con restricciones de recursos.

Para apoyar el proceso de selección de características, Chen [101] propone una estrategia orientada a la selectividad para acortar las secuencias de decisión. La selectividad de una características se refiere al valor que indica el impacto de seleccionar esa característica en la selección y eliminación de otras características. La selectividad se determina por dos factores: (1) el número de productos comunes y (2) el número de características que serán automáticamente

seleccionados o eliminados mediante la selección de esa característica debido a la propagación de restricciones. Para calcular la selectividad de una característica, transforman un modelo de características en una expresión algebraica denominada descripción de elección (choice description).

En [74] se presenta un enfoque híbrido que combina sistemas de inferencia difusa y metaheurísticas multi-objetivo NSGA-II para apoyar la gestión de productos mediante la generación de carteras de productos, basadas en segmentos de usuarios y en el costo de desarrollo de los activos de la LPS. Los sistemas de inferencia difusa se utilizan para generar el costo de desarrollo de un activo mediante el acoplamiento, el número de líneas de código, la complejidad ciclomática y para estimar la calidad de los productos generados por el módulo de optimización del enfoque. La metaheurística NSGA-II se utiliza para buscar productos que minimicen el costo y maximicen la relevancia de los productos candidatos.

En [83] proponen un algoritmo genético para manejar múltiples objetivos en conflicto en la generación de pruebas para LPS. Esto debido a que la prueba completa de un producto es una tarea costosa y toma demasiado tiempo y esfuerzo. Los investigadores requieren obtener un conjunto de casos de prueba pertinentes a elegir como representante del caso de prueba y de esta forma eliminar los casos de prueba redundantes y aumentar la eficiencia de las pruebas.

En [96] proponen una algoritmo de optimización multi-objetivo IVEA para optimizar la selección de características con requerimientos funcionales y no funcionales considerando las relaciones entre estas características.

En [76] se emplean los algoritmos genéticos para minimizar tres inconsistencias del modelo de características, es decir, obligatorias, inclusivas y exclusivas/alternativas. Presentan los desafíos para desarrollar un algoritmo genético para la optimización de la selección de características de una LPS con respecto a la minimización de inconsistencias. Los desafíos son: 1) evitar la des-selección de todas las características con inconsistencias y 2) cómo asignar priori-

dades / pesos a diferentes tipos de inconsistencias.

En [82], los autores presentan un enfoque para la ubicación de características en las LPS basadas en modelos denominado GA-FL (Genetic Algorithm to Feature Location). La ubicación de características (FL) se conoce como el proceso de encontrar el conjunto de artefactos de software que realizan una característica particular, es decir implementan una funcionalidad específica.

Tabla 2.1: Análisis comparativo de trabajos relacionados

Propuesta	**Enfoque**	**Técnica de SBSE**	**Restricción**	**Función objetivo**
[88]	LPS	Lógica Difusa	No	Gestión de prioridades
GAFES [73]	LPS	Algoritmo genético	Si	Minimizar los recursos consumido
ACOFES [78]	LPS	Colonia de hormigas	Si	Minimizar el consumo de tiempo
[74]	LPS	Sistemas de inferencia difusos y optimización multi-objetivo.	Si	Minimizar el costo y maximizar la relevancia de los productos candidatos
IVEA [96]	LPS	Optimización multi-objetivo	Si	Preferencia y rendimiento
[76]	LPS	Algoritmo genético	No	Minimizar inconsistencias del modelo de características.
MPL-FES	MPL	Optimización multi-objetivo y algoritmos genéticos	Si	Minimizar el costo de desarrollo y maximizar la compatibilidad y reutilización de las características.

La principal diferencia de los trabajos mencionados anteriormente es que la actual propuesta (MPL-FES) se centra en el diseño de una MPL, en la cual los aspectos comunes y variables se rediseñan en activos reutilizables para ampliar el alcance de las n LPS. Si bien existe una estrecha relación entre todas las propuestas, es importante tener en cuenta que la propuesta aborda la

problemática de configuración de productos de software bajo un enfoque de optimización multi-objetivo compensando el antagonismo entre el criterio económico y los dos criterios técnicos (compatibilidad y reutilización) facilitando así la toma de decisiones al momento de elegir las posibles configuraciones. Además, el modelo garantiza la satisfacción de las restricciones del modelo de características MPL (obligatorio, opcional y/o alternativas) durante el proceso de configuración del producto.

2.2. Desafío 2: Reutilización de LPS

La Ingeniería de Líneas de Productos de Software permite el desarrollo eficiente de software a medida mediante artefactos reutilizables. Los profesionales reconocen que existe gran potencial para reutilizar LPS en otras LPS (MPL). A continuación, se presentan los trabajos identificados en la literatura que abordan la reutilización de LPS en MPL.

En [56] presentan una extensión al actual modelado de LPS basándose en un diagrama de clases para describir instancias de LPS y dependencias entre ellas. Utilizan la especialización de LPS para reutilizar configuraciones de LPS entre diferentes composiciones de LPS. El objetivo de la propuesta es conectar el modelado de dominio y la implementación de dominio; mientras que los modelos de características describen las características de una LPS, las instancias de las LPS describen la composición de LPS.

En [62] proponen interfaces para diferentes niveles de abstracción en múltiples etapas del proceso de desarrollo de MPL tales como:

1. **Interfaz del modelo de variabilidad:** se ocupa de la distribución de la funcionalidad provista por un modelo de variabilidad que se necesita en otro modelo. Representa el acuerdo entre el uso y la reutilización de una LPS.

2. **Interfaz sintáctica:** Interfaz de Programación de Aplicaciones (API) que contiene variabilidad. Representa una vista de los artefactos de código reutilizables de una LPS sin detalles de implementación.

3. **Interfaz de comportamiento:** acuerdo (DbC) sobre el comportamiento de los diferentes métodos para garantizar la correcta comunicación entre las múltiples LPS.

4. **Interfaz de propiedades no funcionales:** describe las propiedades no funcionales de una LPS en la que confían otras LPS.

Savolainen y otros autores [47] basándose en la experiencia industrial con cuatro empresas exploran las características de los contextos en los que las MPLs son una estrategia de desarrollo viable y proponen un marco de trabajo de opciones para el desarrollo de la plataforma:

1. **LPS independientes desde el punto de vista técnico.** Generalmente esto ocurre por dos razones: 1) las LPS resultan de fusiones y adquisiciones en las que se adquiere una nueva LPS complementaria fuera de la empresa y 2) la empresa decide construir una nueva LPS por separado. Esto ocurre por problemas significativos con las LPS actuales o la necesidad de establecer un nuevo sitio totalmente independiente sin dependencias de la organización actual.

2. **Reutilización de procesos y herramientas.** Las herramientas son un punto de partida natural para lograr efectivamente la reutilización del contenido entre las LPS. Esto incluye la administración de la configuración, los requisitos, colaboración y herramientas de documentación.

3. **Reutilización de especificaciones externas.** La reutilización de requisitos a través de LPS también reduce el costo de crear los activos de la LPS. Los requisitos comunes permiten tomar decisiones explícitas sobre lo que es común y lo que varía entre los productos y las LPS. Esto es significativo, ya que la capacidad de definir y mantener el alcance adecuado para cada LPS contribuye en gran medida al éxito de la empresa. Una amenaza clave al tener múltiples LPS es que comienzan a competir entre sí. Para reducir la competencia

entre LPS es recomendable tener una estructura de requisitos común que contenga todos los requisitos de los productos independientemente de la LPS a la que pertenezcan.

4. **Reutilización de especificaciones internas**. Los requisitos y las pruebas examinan el sistema desde el punto de vista externo, pero existen opciones adicionales para reutilizar especificaciones cuando se considera la estructura interna del sistema. Dentro de las opciones para reutilizar las especificaciones internas se encuentran: estandarizar algunas interfaces de las LPS, derechos de propiedad intelectual, especificaciones de los componentes y la generación de una arquitectura de referencia común para varias LPS.

5. **Reutilización de la implementación.** También es posible compartir los activos de las implementaciones actuales, especialmente las de las características comunes. Sin embargo, un problema frecuente en la reutilización de características comunes a través de los productos es que deben adherirse a diferentes restricciones de desempeño y limitaciones que hacen que el uso de la misma implementación sea difícil.

    - Enfoque componentes compartidos
    - Middleware común
    - Plataforma de computación común
    - Enfoque combinado

6. **Reutilización del ecosistema**. Mediante un ecosistema de software es posible utilizar una arquitectura de referencia, comprar un diseño, recopilar el software y comprar la integración, pruebas y fabricación. El inconveniente es que el producto será muy similar a otros productos que se han creado utilizando este enfoque. Sin embargo, es posible crear una LPS que compita con éxito contra el ecosistema dominante.

2.3. Desafíos 6 y 7: Interoperabilidad y Arquitecturas de Referencia

En [1] se enumeran las principales perspectivas de investigación de las arquitecturas de referencia en las MPL:

- Resultan útiles para establecer interfaces entre las diversas LPS que integran una MPL. Cada LPS y sus productos se desarrollan sobre la base de una arquitectura de referencia para facilitar la interoperabilidad e integración con otras LPS y sus productos, resultando en una MPL.

- Se utilizan para establecer un medio común de comunicación entre diversas organizaciones que participan en el desarrollo de una MPL. Basándose en una arquitectura de referencia, una organización conoce exactamente cuáles son las interfaces de sus sistemas y cómo se comunican con otros sistemas para constituir un SoS.

- Evolución y refinamiento de arquitecturas de referencia existentes para la MPL. Es necesario identificar las arquitecturas de referencia existentes para identificar su capacidad para satisfacer el enfoque de MPL. Una MPL conduce nuevas necesidades como modularidad adecuada y promover la interoperabilidad. Si estas arquitecturas no son adecuadas para la MPL y SoS, es necesario adaptarlas a este nuevo contexto.

- Es necesario construir nuevas arquitecturas de referencia para utilizarse en el contexto de MPL. Los dominios que desarrollan SoS son candidatos para adoptar MPLs y construir arquitecturas de referencia. Como ejemplos de dominios: sistemas de información, sistemas embebidos, automotores, militares y comerciales. Además, éstas Arquitecturas de Referencia son factibles de utilizarse como un estándar para desarrollar sistemas para ese dominio.

Tekinerdogan y otros [45] proponen un enfoque para el análisis de la conformidad (trazabilidad) de la MPL basándose en el modelado de reflexión. Es decir, permite comprobar la

consistencia de la arquitectura de la MPL. La consistencia de la arquitectura implica que los elementos de diseño de la arquitectura se mapean a los elementos de implementación. En caso de que las relaciones entre la arquitectura y la implementación no correspondan, entonces se llaman violaciones de arquitectura. Si las relaciones que están presentes en la arquitectura también se encuentran en la implementación, entonces es una relación convergente. En caso de que la relación de arquitectura no esté presente en la implementación, entonces se llama una relación de ausencia. Las relaciones de ausencia ocurren durante el desarrollo inicial del sistema en el que se define la arquitectura pero la implementación no está lista todavía. En las primeras fases del desarrollo estas relaciones de ausencia son un asunto menor. Finalmente, si la implementación incluye una relación que no está presente en la arquitectura, entonces se denomina relación de divergencia. Las violaciones arquitectónicas surgen por relaciones de ausencia o divergencia.

En [44] se presenta el enfoque denominado Archample (Architectural Analysis Approach for Multiple Product Line Engineering) para el análisis de la arquitectura en el contexto de Ingeniería de MPLs. A diferencia de los enfoques existentes de análisis de arquitecturas, Archample se enfoca en el análisis de la arquitectura de una MPL en particular (Aselsan REHIS). El objetivo de Archample es apoyar la decisión sobre si es necesario utilizar una arquitectura MPL y asimismo evaluar diferentes alternativas de descomposición de la arquitectura MPL. Archample introduce puntos de vista arquitectónicos *(architectural viewpoints*) para el modelado y documentación de MPLs.

En [22], los autores estudian cómo evaluar un Sistema de Sistemas que consiste en varias LPS que cooperan. La intención es evaluar la arquitectura de todo el SoS con el fin de satisfacer un nuevo conjunto de requisitos transversales. Describen las experiencias y prácticas de realizar una evaluación de la arquitectura del SoS en la industria: el SoS en cuestión es un conjunto de LPS cuyos productos se utilizan para crear la red de telecomunicaciones All-IP 3G. Los resultados indican que hay diferencias significativas en la evaluación de la arquitectura del SoS en comparación con las evaluaciones tradicionales dirigidas a sistemas individuales. Las dos diferencias principales que afectan a la evaluación de la arquitectura son la heterogeneidad en

los niveles de madurez de los sistemas individuales, es decir, LPS, y la opción de que, en lugar de evaluar cada LPS individualmente, es posible mover las responsabilidades de una LPS a otra para satisfacer los requisitos a nivel de SoS.

2.4. Desafío 10: Configuración de la MPL

Actualmente, los enfoques de configuración se basan en el ingeniero de aplicaciones para interpretar adecuadamente los requisitos del cliente y para tomar las decisiones pertinentes. La configuración de la MPL es un proceso de colaboración entre diferentes personas y equipos con conocimientos diferentes con respecto al dominio, notaciones y lenguajes utilizados en el proceso. Por ejemplo, los problemas surgen porque se requiere tomar las decisiones adecuadas para asignar a las partes interesadas, el orden correcto y el momento adecuado basado en el conocimiento del dominio de las personas.

Rosenmüller y Siegmund [56, 14] extienden el proceso de Ingeniería de Líneas de Productos con el modelado de composición de LPS dependientes que forma parte del proceso de diseño de dominio con el fin de automatizar la configuración de MPLs e integrar el modelado MPL en el proceso de desarrollo de las LPS. El modelo de composición es un elemento para crear una MPL que conecte el modelo de dominio y la implementación de una MPL describiendo para cada LPS qué instancias de otras LPS utiliza. El modelo de composición permite describir las dependencias de todas las LPS y proporciona un medio para configurar automáticamente las LPS de acuerdo con estas dependencias. Los pasos para crear una MPL propuestos por [14] son: 1) crear un modelo de características para una MPL, 2) generar y refinar modelos de composición y 3) obtener un generador para la configuración. En la actualidad, el generador de configuración no admite la redefinición de restricciones de modelos en LPS especializados.

En [68] abordan los desafíos de la configuración y generación de productos en un contexto de LPS a gran escala en el negocio de soluciones. Identifican cuatro desafíos: la necesidad de la configuración descentralizada, la configuración heterogénea, el soporte para la generación

de productos y el apoyo a la Ingeniería de Aplicaciones. Abordan estos desafíos con el enfoque (PLiC, Product line configuration) el cual se asemeja al hecho de que el dominio de una LPS a gran escala se divide en subdominios. Un PLiC encapsula todos los artefactos (activos de configuración, generación e implementación) de un sub-dominio y ofrece interfaces para la configuración y generación. El enfoque PLiC se encuentra en fase de implementación y evaluación prototípica.

En [12] presentan a DOPLER (Decision-Oriented Product Line Engineering for effective Reuse), un enfoque basado en el patrón de "publicador / suscriptor "y una herramienta prototipo (DOPLER Tool Suite) que pretende mejorar la conciencia de múltiples usuarios involucrados en la derivación de productos compartiendo decisiones entre diferentes proyectos de derivación independientemente de los meta-modelos o activos de solución técnica utilizados en los diferentes subsistemas.

En [48, 49, 50] se presenta un marco de trabajo (*framework*) denominado Invar (Integrated view on Varibility) que permite intercambiar (plug-and-play) modelos de variabilidad. Invar facilita el intercambio de modelos heterogéneos de variabilidad durante la configuración del producto (independientemente de las técnicas, notaciones y herramientas utilizadas en la organización) y permite que estos modelos estén disponibles a través de los límites de la organización. "Plugging "se refiere a añadir nuevos modelos de variabilidad a un repositorio compartido. "Playing "se refiere a presentar las opciones al usuario final, lo que le permite configurar el producto requerido. Para este propósito, un modelo de variabilidad se ve como una entidad autónoma que se conecta en el espacio de configuración para proporcionar opciones de configuración. Sin embargo, autónomo no significa necesariamente independiente, porque es posible que los modelos de variabilidad se relacionen entre sí. El enfoque permite el uso de modelos de variabilidad distribuidos a través de múltiples repositorios accediendo a ellos a través de Servicios Web que proporcionan opciones de configuración. Un usuario final trabaja con un front-end para la configuración del producto y utiliza los servicios de configuración sin conocer los detalles sobre los

modelos de variabilidad detrás de los servicios.

En [19] investigan la configuración de múltiples LPS dependientes que constituyen una MPL y presentan tres contribuciones: 1) un conjunto extensible de tipos de dependencia (emergentes y establecidos) para modelar dependencias entre opciones de configuración de diferentes LPS que constituyen una MPL. Las dependencias emergentes son creadas por los usuarios durante la configuración para expresar dependencias informales entre LPS. Las dependencias establecidas se definen analizando las dependencias emergentes existentes entre LPS , 2) un mecanismo de tablero de anuncios denominado tablero de decisión (decision board) para capturar dependencias emergentes. Los usuarios finales utilizan un mecanismo de publicación-suscripción que les notifica acerca de las opciones de configuración para otras LPS. Este mecanismo aumenta la conciencia de los usuarios y al mismo tiempo reúne información sobre posibles dependencias de configuración y 3) una extensión a la herramienta DOPLER Tool Suite que proporciona apoyo a la configuración distribuida de las MPL por múltiples usuarios.

En [102] proponen el ocultamiento de la variabilidad en las especificaciones para la verificación eficiente de LPS dependientes en evolución. Esto se refiere a ocultar detalles innecesarios de una LPS para otras LPS. Extienden el trabajo anterior [62] sobre interfaces de modelos de características e interfaces de contexto de características mediante interfaces de LPS de comportamiento, que ocultan características de los contratos para facilitar el proceso de verificación. Con la ocultación de la variabilidad, eliminan las referencias de las características de los contratos para reducir el esfuerzo de verificación de las líneas de productos dependientes. Discuten dos estrategias para enlazar la variabilidad, configuración falsa y verdader. Es decir, verifican sólo un subconjunto de todas las configuraciones posibles de la LPS reutilizada, descartando algunas configuraciones. En consecuencia, no hay potencial para conectar la mejor configuración de acuerdo a las propiedades no funcionales. Por el contrario, la configuración oculta elimina características sin variabilidad vinculante. Para las tres estrategias de interfaz, no está claro hasta qué punto mejoran la eficiencia.

En [54] presentan un enfoque automatizado apoyado en una herramienta denominada SpineFM para gestionar LPS a través de un modelo de dominio que interrelaciona varios modelos de características de forma consistente. El enfoque transfiere parte del conocimiento del dominio al espacio del problema y apoya la derivación de configuraciones complejas con instanciaciones múltiples y asociaciones de sub-productos. Utiliza un algoritmo de propagación de acciones de configuración para proporcionar un proceso de ordenación libre para la derivación del producto. En [21] presentan un enfoque basado en herramientas (DOPLER Tool Suite) para supervisar los requisitos de un SoS (Sistema de Sistemas) formalizados como restricciones durante la derivación distribuida de productos distribuidos en MPLs.

Czarnecki [103] propone la especialización y la configuración multi-nivel como dos mecanismos diferentes para realizar la configuración escalonada o por etapas. En la configuración multi-nivel, cada etapa tiene su propio conjunto de opciones de configuración que requieren realizarse en esa etapa, mientras que las opciones de configuración en la especialización no se limitan a ninguna etapa en particular. Por último, la creación de modelos de características independientes para diferentes etapas evita la "parálisis de análisis", un error común en el modelado de características, que equivale a un desglose entre los diferentes niveles de abstracción de las características individuales.

2.5. Trabajos relacionados

A continuación, se presenta el panorama general de las Multilíneas de Productos de Software ya que a pesar del interés para los investigadores dentro del área del software y la experiencia en la industria esta área de investigación reporta diversos desafíos que necesitan solventarse.

2.5.1. Enfoques MPL

A continuación, se muestra la revisión bibliográfica de los enfoques existentes que apoyan el desarrollo y la implementación de MPLs clasificados de la siguiente forma:

- **Ingeniería del Dominio:** descomposición de modelos, composición y reutilización de componentes y administración de las dependencias entre subsistemas (Tabla 2.2).
- **Ingeniería de la Aplicación:** configuración, resolución y representación de restricciones (Tabla 2.3)
- **Híbrido:** enfoques que apoyan la Ingeniería del Dominio y la Ingeniería de Aplicación (Tabla 2.4).
- **Organizativo:** administración de los equipos de las Líneas de Productos de Software o interdependencia de unidades de negocio (Tabla 2.5).

Tabla 2.2: Enfoques MPL - Ingeniería del Dominio

Enfoque	Descripción	Herramienta	Referencia
Poblaciones de productos	Conceptos basados en componentes para la gestión de las poblaciones de productos.	Koala Tools	[13, 104, 105, 106]
Establecer la representación de líneas de productos	Representación (jerárquica) de Líneas de Producto como un conjunto.		[107]
Reutilización a través de líneas de productos	Desarrollo de componentes compartidos utilizados en dos o más LPS.		[108]
Línea de Productos Orientada a Servicios (*SOPL, Service-Oriented Product Line*)	Combinación de productos de las LPS utilizando el paradigma de Arquitecturas Orientadas a Servicio (*SOA, Service-Oriented Architecture*). SOPL se refiere a un escenario en el que una LPS consume productos de proveedores de LPS de terceros para producir productos más complejos.		[109]

Continúa en la página siguiente

Tabla 2.2 Enfoques MPL - Ingeniería del Dominio (continuación)

Enfoque	Descripción	Herramienta	Referencia
Modelo de variabilidad contextual	El modelo de variabilidad contextual representa el contexto en el que se espera utilizar un producto derivado de la línea de productos.		[60]
Automatización de Fábricas de Software (*Software Factory Automation*)	Gestión de activos reutilizables a través de diferentes Líneas de Productos de Software basadas en kits específicos del dominio (*DSK, Domain Specific Kits*) y "Meta modelo de activos de software"(*Software Asset Meta Model*).		[110]
Composición de Líneas de Productos de Software	Apoyo a la composición vertical utilizando artefactos de niveles inferiores y composición horizontal compartiendo el mismo entorno.		[111]
Modelo de características independiente del proveedor (*SIFM, Supplier Independent Feature Model*)	El SIFM contiene el super-conjunto de características de todos los proveedores, junto con un punto de variación para distinguir entre proveedores.		[46]
Gestión de la variabilidad composicional (*CVM, Compositional variability management*)	Marco de trabajo (*framework*) para la gestión de la variabilidad composicional basado en el concepto de enlaces de configuración.	CVM	[112, 113]

Continúa en la página siguiente

Tabla 2.2 Enfoques MPL - Ingeniería del Dominio (continuación)

Enfoque	Descripción	Herramienta	Referencia
Fragmentos del modelo	Un fragmento de modelo describe los activos reutilizables y la variabilidad para una parte arbitraria de la LPS (por ejemplo, un conjunto de características, un subsistema o una funcionalidad transversal). Los fragmentos de modelo sirven como la unidad básica de evolución y son creados y mantenidos por equipos individuales que solo están acoplados entre sí. Los fragmentos modelo nunca se utilizan directamente en la derivación del producto. Es posible que evolucionen independientemente y a diferentes velocidades.	DOPLER Tool Suite	[114]
Técnicas de fusión para administrar MPLs	Enfoque composicional para gestionar MPLs, las cuales involucran la fusión automática de modelos de características definidos a través de las LPS.		[8]
Relaciones entre sistemas	Un punto de vista arquitectónico de los S3 (sistemas intensivos de software) que consiste en una taxonomía de relaciones a nivel de sistema.		[115, 116]

Continúa en la página siguiente

Tabla 2.2 Enfoques MPL - Ingeniería del Dominio (continuación)

Enfoque	Descripción	Herramienta	Referencia
MPLs mecatrónicas	Enfoque para distinguir entre el software y el hardware utilizando diversos modelos de características para cada uno. Introducen un nivel de abstracción para líneas de productos complejas que consisten en múltiples modelos de características para establecer un mapeo de modelos de características.	FOCUS prototype	[52]
Dependencias de configuración en MPL	Conjunto extensible de tipos de dependencia (Emergentes y Establecidos) para modelar dependencias entre opciones de configuración de diferentes LPS que constituyen una MPL.	Extensión a DOPLER Tool Suite	[19]
Interfaces en modelos de variabilidad	Interfaces para diferentes niveles de abstracción en múltiples etapas del proceso de desarrollo de una MPL.	Extensión a FeatureIDE	[62]

Continúa en la página siguiente

Tabla 2.2 Enfoques MPL - Ingeniería del Dominio (continuación)

Enfoque	Descripción	Herramienta	Referencia
Modelos de características dependientes (DFM, Dependent Feature Model)	Enfoque de 5 pasos para analizar de forma automática los DFMs: 1. Análisis del modelo 2. Comprobación sintáctica 3. Traducción del modelo 4. Comprobar satisfacibilidad 5. Otros análisis Además, introduce análisis especiales para DFMs, como: características dependientes-muertas, características dependientes-falsas-opcionales, características principales e instancias centrales, número o productos y variabilidad de instancias	VeAnalyzer	[58]
Interfaces multinivel para el análisis de MPLs	El concepto de interfaces multinivel consiste en un conjunto de interfaces que encapsulan modelos de variabilidad, la interfaz de programación y el comportamiento en tiempo de ejecución de las LPS.	Extensión de FeatureIDE	[59]
Enfoque orientado al actor	Proceso de mapeo de arquitectura de referencia, arquitectura de LPS y arquitectura de sistemas de una MPL. El actor actúa como el conector de varias MPLs.		[55]

Continúa en la página siguiente

Tabla 2.2 Enfoques MPL - Ingeniería del Dominio (continuación)

Enfoque	Descripción	Herramienta	Referencia
Modelo MPL	Modelo formal para MPL basado en el concepto de LPS dependientes y el subtipo para: a) capturar la noción de relación entre diferentes componentes, b) describir diferentes sistemas compuestos reales y c) apoyar la construcción de herramientas para el diseño, mantenimiento y análisis de sistemas compuestos.		[11]
Modelo formal para MPL	Modelo formal para MPLs que abarca el modelo de características, la base de artefactos y el conocimiento de configuración. El modelo se construye alrededor de los conceptos de firmas de LPS, LPS dependientes y la composición de LPS.		[43]

Tabla 2.3: Enfoques MPL - Ingeniería de Aplicación

Enfoque	Descripción	Herramienta	Referencia
Configuración por etapas y multi-nivel	Configuración realizada en etapas en un subconjunto de características organizadas en múltiples niveles. La configuración multi-nivel evita el problema de “parálisis de análisis”en el modelado de características, lo que equivale a una descomposición entre los distintos niveles de abstracción de las características individuales.		[103]

Continúa en la página siguiente

Tabla 2.3 Enfoques MPL - Ingeniería de Aplicación (continuación)

Enfoque	Descripción	Herramienta	Referencia
Configuración colaborativa y coordinada de productos	Definición de decisiones y roles para la configuración colaborativa de productos.	Prototipo	[66]
LPS Dependientes / Modelos de composición	Enfoque para automatizar la configuración de MPLs basado en modelos de composición.		[56, 14]
Flujos de trabajo de configuración de características	Los flujos de trabajo definen tareas y roles para la configuración de un modelo de características en etapas.		[117]
Product line configuration (PLiC)	Un PLiC encapsula todos los artefactos (activos de configuración, generación e implementación) de un sub-dominio y ofrece interfaces para la configuración y generación.	Extensión de Eclipse	[68, 118, 119]
DOPLER (Decision-Oriented Product Line Engineering for effective Reuse)	Enfoque basado en el patrón de "publicador/suscriptor" y una herramienta prototipo que pretende mejorar la conciencia de múltiples usuarios involucrados en la derivación de productos compartiendo decisiones entre diferentes proyectos de derivación independientemente de los meta-modelos o activos de solución técnica utilizados en los diferentes subsistemas.	Extensión a DOPLER Tool Suite	[67, 12]

Continúa en la página siguiente

Tabla 2.3 Enfoques MPL - Ingeniería de Aplicación (continuación)

Enfoque	Descripción	Herramienta	Referencia
Invar (Integrated view on Variability)	Un marco de trabajo (*framework*) que permite "plug-and-play"modelos de variabilidad. Invar facilita el intercambio de modelos heterogéneos de variabilidad durante la configuración del producto (independientemente de las técnicas, notaciones y herramientas utilizadas en la organización) y permite que estos modelos estén disponibles a través de los límites de la organización.	Prototipo Invar	[48, 49, 50]
Composición dirigida por modelos	Enfoque que permite expresar una relación de uso entre diferentes LPS. Esto permite controlar la consistencia del sistema, propagar las opciones del usuario y limitarlas a configuraciones válidas.		[120]
Plataforma para MPL	Formulación de opciones para la gestión de una MPL: 1) Líneas de productos independientes, 2) Reutilización de procesos y herramientas, 3) Reutilización de la especificación externa, 4) Reutilización de la especificación interna, 5) Reutilización de la implementación y 6) Reutilización del ecosistema.		[47]

Continúa en la página siguiente

Tabla 2.3 Enfoques MPL - Ingeniería de Aplicación (continuación)

Enfoque	Descripción	Herramienta	Referencia
Monitorización de requisitos SoS	Enfoque para detectar violaciones de múltiples requisitos de sistemas durante la configuración de sistemas individuales y proporcionar retroalimentación inmediata a los configuradores implicados.	DOPLER Tool Suite	[21]
Configuraciones complejas en LPS	Enfoque automatizado para gestionar LPS a través de un modelo de dominio que interrelaciona varios modelos de características de forma consistente. Utiliza un algoritmo de propagación de acciones de configuración para proporcionar un proceso de ordenación libre para la derivación del producto.	SpineFM	[54]
Impacto del cambio en MPLs	Un enfoque para calcular el impacto de un cambio de característica en las configuraciones existentes de una MPL, utilizando la información parcial de las restricciones entre los modelos de características.	Neo4JFM	[65]
Ocultamiento de la variabilidad en contratos para LPS dependientes	La ocultación de la variabilidad se refiere a ocultar detalles innecesarios de una LPS para otras LPS. Este trabajo reduce el esfuerzo de verificación de las líneas de productos dependientes en evolución mediante la eliminación de referencias de características de los contratos.	FeautureIDE	[102]

Tabla 2.4: Enfoques MPL - Híbrido

Enfoque	Descripción	Herramienta	Referencia
Product line bundles (PLiB)	Componente autónomo que encapsula un modelo de variabilidad, un manifiesto (tabla de contenidos y metadatos) y artefactos adicionales de una LPS (políticas organizacionales, dependencias a otros PLiBs y una fecha de caducidad) para derivar productos. PLiBs permiten configurar diferentes LPS en una MPL.	DOPLER Tool Suite	[121, 57, 20]
OOFM - A Feature Modeling Approach	Aplicación de la técnica de Modelado de Características Orientada a Objetos (OOFM, Object-Oriented Feature Modeling) para desarrollar MPLs y LPS dinámicas (DSPLs). La idea es combinar recursos OOFM con un estilo arquitectónico Modelo Vista Controlador (MVC).		[122]
CoDiM (Constraint Checking in Distributed Configuration of MPLs)	Enfoque apoyado en una herramienta para definir y controlar restricciones en la configuración distribuida de un SoS. CoDiM soporta plantillas de restricciones que son parametrizadas para permitir su reutilización en diferentes configuraciones de MPLs.	Extensión a DOPLER Tool Suite	[123]

Continúa en la página siguiente

Tabla 2.4 Enfoques MPL - Híbrido (continuación)

Enfoque	Descripción	Herramienta	Referencia
Explotar el conocimiento del dominio en MPLs	Necesidad de modelado orientado al dominio para apoyar la definición de MPLs basándose en múltiples modelos de características. Los tres retos principales relacionados con el modelado de dominio de una MPL son: 1. La gestión de la relación a nivel de dominio. 2. El concepto de configuración relacionada con el dominio. 3. El apoyo al proceso de configuración del dominio.		[124]
Gestión del ciclo de vida de las MPLs	Enfoque que apoya la gestión del ciclo de vida de las MPLs y utiliza PLIBs como bloques de construcción de la infraestructura MPL.	DOPLER MPL Editor	[20]
Análisis de la conformidad de la arquitectura	Enfoque para comprobar la consistencia de la arquitectura dentro del contexto de la ingeniería de MPL basado en el modelado de la reflexión.		[45]
Architectural Analysis Approach for Multiple Product Line Engineering (Archample)	Enfoque para el análisis de la arquitectura en el contexto de Ingeniería de MPLs. Archample introduce puntos de vista arquitectónicos (architectural viewpoints) para el modelado y documentación de MPLs.		[44]

Continúa en la página siguiente

Tabla 2.4 Enfoques MPL - Híbrido (continuación)

Enfoque	Descripción	Herramienta	Referencia
Variabilidad del modelo de datos	Método para desarrollar un modelo de características universal de la MPL obtenido a partir de la integración de los modelos de características de las LPS individuales, incorporando las interdependencias de datos entre las características. Asimismo, un método para generar el modelo de datos de la MPL utilizando una técnica orientada deltas.		[61]
FORCE (*Feature-ORiented Component Engineering*)	Un lenguaje de modelado y un entorno de herramientas para apoyar el modelado de características multipropósito y multi-nivel, incluyendo el mapeo al código base.	Extensión a FeatureIDE (FORCE tool)	[125]
MPL basada en SOA	Enfoque basado en modelos, de arriba hacia abajo (*top-down*), formal y de extremo a extremo (*end-to-end*) orientado a servicios (SOA) basado en el paradigma de MPL. La idea principal es integrar las técnicas de LPS y MPL con SOA para gestionar e interrelacionar la variabilidad de los Consumidores de Servicio (CSs) y los Proveedores de Servicios (PSs).	MSPL4SOA	[126]

Tabla 2.5: Enfoques MPL - Organizacional

Enfoque	Descripción	Herramienta	Referencia
Familias jerárquicas de productos	Categorías de productos cubiertas por líneas de productos gestionadas por unidades separadas de negocio: departamento de desarrollo, unidades de negocio, unidad de ingeniería del dominio y unidades jerárquicas de ingeniería del dominio.		[127]
LPS Composicionales	Equipos autónomos responsables de Líneas de Productos de Software.		[128]

En la actualidad, se reportan diversos enfoques y propuestas que apoyan el desarrollo de software utilizando MPLs. Sin embargo, los estudios demuestran una serie de desafíos y diversas áreas de oportunidad en este campo de investigación ya que el alcance de la MPL depende principalmente de factores como el tamaño y complejidad de los sistemas, evolución de las LPS, cambio en los requisitos por el mercado o tecnología y capacidades del equipo de desarrollo.

Entre los desafíos identificados destacan: inconsistencias en la configuración de activos, dificultades en la interoperabilidad de LPS, creación de una arquitectura de referencia para definir partes comunes y diferentes entre líneas de productos de software, se requieren técnicas que faciliten la reutilización y el análisis modular, soporte para la gestión de la variabilidad y la configuración de productos en una MPL, modelar MPLs, integrar modelos de composición en una MPL, adaptación y extensión de herramientas LPS en el contexto de MPL, entre otros.

Capítulo 3

Aplicación de la metodología

Este capítulo describe las actividades realizadas para resolver la problemática planteada en la investigación actual.

3.1. Metodología de la investigación

La metodología empleada para el desarrollo de la presente investigación se describe a continuación:

1. Análisis del estado del arte de las Líneas de Productos de Software y las Multilíneas de Productos de Software.
2. Identificación de los artefactos que intervienen en el desarrollo de una MPL.
3. Identificación y análisis de las herramientas que apoyan el desarrollo de una MPL.
4. Identificación de las técnicas de Ingeniería de Software Basada en Búsqueda para resolver problemas en el ciclo de vida de las LPS.
5. Definición de las estrategias de composición de LPS para la implementación de una MPL.
6. Análisis de los enfoques disponibles en la literatura para representar las Arquitecturas de Referencia en las MPL.

7. Definición de un caso de estudio para el diseño e implementación del modelo matemático y la MPL en el área de Inmótica.

8. Diseño y solución de un modelo matemático para optimizar el proceso de configuración de productos en una MPL.

9. Desarrollo de los artefactos necesarios para la implementación de la MPL.

3.2. Análisis del estado del arte

La revisión del estado del arte comprende la revisión y el análisis de los artículos identificados y relacionados directamente con las Líneas de Productos de Software y Multilíneas de Productos de Software. Este análisis se identifica en el capítulo 2.

3.3. Identificación de los artefactos para el desarrollo de la MPL

Una plataforma en el contexto del software es una colección de artefactos reutilizables. Estos artefactos deben reutilizarse de forma consistente y sistemática para construir aplicaciones. Los artefactos reutilizables abarcan todo tipo de artefactos de desarrollo de software.

Un **artefacto de desarrollo** es la salida de un sub-proceso de Ingeniería del Dominio o Aplicación. Los artefactos de desarrollo abarcan requisitos, arquitectura, componentes y pruebas. A continuación, se describen los artefactos identificados en la literatura necesarios para apoyar el desarrollo de Multilíneas de Productos de Software.

- Una **arquitectura de referencia** es un tipo de arquitectura de software que captura la esencia de las arquitecturas de sistemas de un dominio dado, es decir, abarcan el conocimiento y la experiencia sobre cómo estructurar arquitecturas de software de los sistemas de ese dominio. Su propósito es guiar el desarrollo, estandarización y evolución de los sistemas de tal dominio o dominios cercanos [129, 1]. Ejemplos de arquitectura de referencia

son: Autosar (AUTomotive Open System Architecture Open Systems for the Automotive sector), UniversAAL (UNIVERsal open platform and reference Specification for Ambient Assisted Living) y Continuous. La Figura 3.1 ilustra el panorama que implica la relación entre las Arquitecturas de referencia y las MPLs. En esta figura, se observa que es posible utilizar una o más arquitecturas de referencia como base de una MPL.

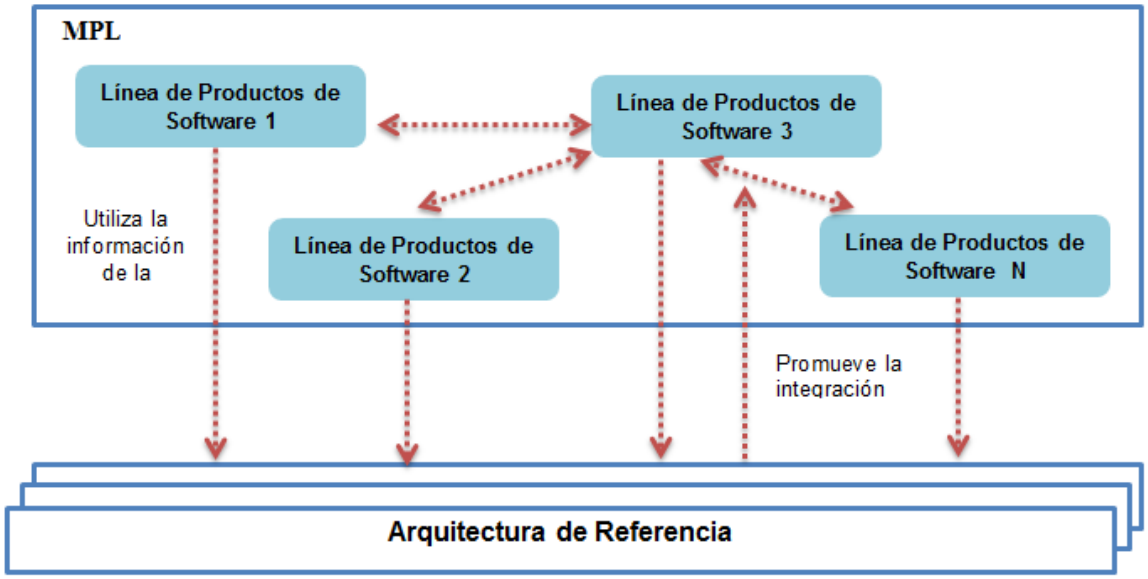


Figura 3.1: Relación entre Arquitectura de Referencia y una MPL [1]

- Una **configuración compleja** es una composición de instancias de elementos del dominio enlazadas de acuerdo con las relaciones definidas por la representación de dominio [54].

- Un **generador** o **configurador** es utilizado para derivar las configuraciones de todas las instancias de LPS de una MPL. El generador de configuraciones proporciona una interfaz gráfica de usuario que solicita a un usuario la selección de características de todas las instancias de LPS necesarias para un escenario de MPL. El generador comprueba las violaciones de las restricciones del modelo y crea las instancias de las LPS requeridas para una configuración de MPL [14].

- El **modelo de características dependiente** (*DFM, Dependent feature model*) es una extensión del modelo de características que permite construir cualquier modelo de caracte-

rísticas de una notación de modelado de características dada (por ejemplo, características opcionales, alternativas, obligatorias, restricciones). Además, existe la posibilidad de que este modelo contenga referencias o dependencias a otros modelos de características (dependientes) [58].

- **MPL-Feature Model (*Multiple Product Line Feature Model*)** [60] es un modelo de características centrado en los datos que contiene todas las características de las LPS y sus interrelaciones. La característica fundamental es la MPL y sus sub-características son los modelos de características de las familias de productos en las MPLs.

- **Variabilidad del contexto** es la variabilidad del entorno de un producto [60].

- El **modelo de variabilidad contextual** se combina con un modelo de características para modelar una MPL utilizando la variabilidad del contexto. Este modelo captura la variabilidad y los aspectos comunes en los requerimientos que se originan en diferentes contextos, como diferentes tipos de producto, regiones geográficas y clientes.

- La **matriz de requisitos de aplicaciones de la MPL** representa los requisitos comunes y variables para todas las aplicaciones (o productos) de las LPS que integran la MPL en una matriz. Generalmente, la columna izquierda de la matriz muestra los requisitos de las aplicaciones consideradas en las LPS. Las aplicaciones se muestran en la fila superior, mientras que el cuerpo de la matriz proporciona información sobre qué requisitos son obligatorios para un determinado producto.

3.4. Identificación de herramientas MPL

Para facilitar el diseño (modelo de características), configuración de productos y desarrollo de la MPL, se identificó y analizó el potencial que ofrecen las herramientas de LPS para las MPL. Asimismo, se adquirieron los fundamentos necesarios para su correcto funcionamiento. Estas herramientas reportan el uso de diferentes enfoques para implementar LPS como:

- **Programación Orientada a Características** (*FOP, Feature-Oriented Programming)*: es un paradigma para la construcción, personalización y síntesis de sistemas de software a gran escala.

- **Programación Orientada a Aspectos** (*AOP, Aspect-Oriented Programming*): técnica de programación que provee al programador de mecanismos adecuados para separar componentes y aspectos, unos de otros, permitiendo su abstracción y composición para producir todo el sistema. La POA se adopta como tecnología de implementación para la derivación de productos a nivel de programa.

- **Programación Orientada a Deltas** (*DOP, Delta-Oriented Programming*): es un paradigma de programación diseñado para implementar LPS, basado en el concepto de programas deltas [130]. En DOP, la implementación de una LPS se divide en un módulo core y un conjunto de módulos delta. El módulo core comprende un conjunto de clases que implementan un producto completo para una configuración válida de características. Los módulos delta especifican los cambios que se aplicarán al módulo core con el fin de implementar otros productos. Un módulo delta permite agregar, remover o modificar clases a la implementación de un producto. El objetivo de DOP es mitigar las restricciones de la Programación Orientada a Características y proporcionar un lenguaje de programación expresivo y flexible para implementar LPS.

Las herramientas disponibles en la literatura que soportan el desarrollo MPL se describen en la Tabla 3.1.

Tabla 3.1: Herramientas MPL

Herramienta	Generador	Lenguaje de programación	Herramientas relacionadas	Formalismos de FM
FeatureIDE [131]	Preprocesador, FOP, AOP, DOP	C, C#, Java, JML, Haskell, XML, Fuji, AspectJ, DeltaJ, DeltaEcore	AHEAD, FeatureC++, FeatureHouse, CPP, Antenna, Munge, DeltaEcore, SPLCATool, TypeChef, Colligens, FORCE tool	The guidsl Tool, S.P.L.O.T., Velvet, fmp: Feature Modeling Plug-in, SPL Conqueror
EASy-Producer [132]	Preprocesador	Java	Xtext	IVML and VIL

Continúa en la página siguiente

Tabla 3.1Herramientas MPL(continuación)

Herramienta	Generador	Lenguaje de programación	Herramientas relacionadas	Formalismos de FM
MutiDeltaJ [133]	DOP	Java	DeltaJ [130]	
DOPLER [67]		Java	DecisionKing, Project-King, ConfigurationWizard	DoplerVML
Invar prototype	Web Services	Java/J2EE		
VeAnalyzer [58]				Velvet
SpineFM [54]		Java	FAMILIAR DSL, Eclipse Modeling Tool	SPLOT, FeatureIDE, TVL and FML
Neo4JFM [65]		Java	FeatureIDE, Neo4j	
MSPL4SOA [126]		Java	FAMILIAR tool	

FeatureIDE es un entorno de desarrollo integrado (*IDE, Integrated Development Environment*) basado en Eclipse que soporta todas las fases de Desarrollo de Software Orientado a Características para el desarrollo de Líneas de Productos de Software: análisis del dominio, implementación del dominio, análisis de requisitos y generación de software. Integra diferentes técnicas de implementación de LPS como: FOP, AOP, DOP y preprocesadores [131, 134, 135, 136]. FeatureIDE está abierto para extensiones y posee soporte para varios lenguajes.

EASy-Producer (EASy stands for Engineering Adaptive Systems) [106, 53, 137, 23] es una herramienta prototípica para el desarrollo de grandes y complejas Líneas de Productos de Software y ecosistemas enriquecidos en variabilidad. Este plug-in de Eclipse admite los siguientes enfoques: configuración de productos en varias etapas, modelado de variabilidad multidimensional y MPL [132]. Además, EASy-Producer integra técnicas para la reducción de la complejidad, composición y derivación de los productos. También, proporciona soporte de dos formas: vistas gráficas para usuarios no expertos y vistas de modelado textual para usuarios expertos. El enfoque textual depende de dos DSL:

1. El lenguaje de modelado de variabilidad integrada (IVML) para la definición de la variabilidad a través del ecosistema de software.

2. El lenguaje de implementación de variabilidad (VIL) para el proceso de derivación del producto.

MultiDeltaJ es una extensión de DeltaJ que se encuentra en desarrollo para apoyar la Programación Orientada a Deltas de MPLs. Es un lenguaje de programación para MPLs orientadas a Deltas que permite obtener MPLs por reutilización de grano fino de LPS orientadas a deltas. Una MPL orientada a deltas es una LPS orientada a Deltas que utiliza (importa) otras LPS [133]. Este enfoque abarca el espacio del problema (modelo de variabilidad), espacio de la solución (artefactos de código) y el espacio de configuración (generación de productos).

DOPLER Tool Suite [67] se implementa como un conjunto de plug-ins de Eclipse y consta de tres herramientas:

1. **DecisionKing** es una herramienta de modelado y gestión de la variabilidad.
2. **ProjectKing** guía y soporta el proceso de derivación y ventas del producto.
3. **ConfigurationWizard** apoya la derivación y personalización de productos de una línea de productos de software.

Invar es una implementación prototípica basada en servicios Web que apoya la integración de tres enfoques de modelado de la variabilidad: modelado de características, modelado de variabilidad ortogonal (OVM) y modelado de decisiones [48, 49, 50].

VeAnalyzer es una herramienta de línea de comandos para el análisis automatizado de modelos de características dependientes (DFMs) especificados en VELVET [58].

Neo4JFM es una herramienta prototípica diseñada para evaluar el impacto de un cambio de características en las configuraciones existentes de una MPL [65].

MSPL4SOA es una herramienta que combina las MPL con el paradigma de las Arquitecturas Orientadas a Servicios (SOA). El objetivo de MSPL4SOA es diseñar una MPL que incluya

dos LPS dependientes, denominados LPS_{PS} respectivamente, para los Proveedores de Servicios (*PSs*) y los Consumidores de Servicio (*CSs*). Estas LPS se utilizan para generar PSs y CSs válidos y consistentes. La herramienta MSPL4SOA se basa principalmente en las tecnologías Java EE y la herramienta FAMILIAR. Esta herramienta se compone de dos generadores que permiten generar PSs y CSs personalizados, válidos y consistentes, es decir: spgenerator y scgenerator [126].

SpineFM [54] es una cadena de herramientas que permite diseñar LPS para crear configuraciones compuestas. Además, esta herramienta utiliza un algoritmo de propagación de acciones de configuración para proporcionar un proceso de ordenación libre para la derivación del producto. En SpineFM, una LPS se define por:

1. Un modelo de dominio que representa, a nivel de usuario final, los conceptos de la familia de software, sus multiplicidades y cómo estos conceptos están interrelacionados.

2. Modelos de características (FMs) que capturan los aspectos en común y las variabilidades de cada concepto, también al nivel del usuario final.

3. Restricciones entre estos modelos de características.

3.4.1. Modelación de la variabilidad de una MPL

En la literatura, se identifican lenguajes que permiten combinar y reutilizar modelos de características de LPS para modelar la variabilidad de una MPL. A continuación, se describen los lenguajes disponibles para el modelado de la variabilidad de MPLs.

- **TVL (*Textual Variability Language*)** es un lenguaje de modelado de características basado en texto [138].

- **Velvet** es un lenguaje basado en la sintaxis de TVL que permite a los ingenieros del dominio: (1) describir dimensiones de variabilidad en modelos separados, (2) componer modelos de variabilidad utilizando tres mecanismos diferentes: herencia, superposición y

agregación y (3) describir configuraciones parciales [33]. Sin embargo, Velvet no soporta el modelado de características basado en cardinalidad.

- **IVML (*Integrated Variability Modeling Language*)** es un lenguaje de modelado de variabilidad y configuración para describir configuraciones de ecosistemas de software enriquecidos con variabilidad [24]. Este lenguaje forma parte del conjunto de herramientas EASy-Producer.

- **FAMILIAR (*for FeAture Model scrIpt Language for manIpulation and Automatic Reasoning*)**, es un lenguaje ejecutable que apoya la manipulación y el razonamiento sobre los modelos de características. FAMILIAR se desarrolló con el lenguaje Java utilizando Xtext, un marco de trabajo (*framework*) para el desarrollo de DSL (*Domain-Specific Language*). Este lenguaje se utiliza para apoyar la gestión de modelos de características en una MPL [9].

- **DoplerVML** es un lenguaje de modelado integrado en Dopler y basado en modelos de decisión para definir líneas de productos. Dopler apoya el modelado del espacio del problema utilizando modelos de decisión y definición del espacio de la solución utilizando modelos de activos los cuales representan tipos arbitrarios de activos reutilizables [139].

3.4.2. Selección de herramientas

Después del análisis de las herramientas disponibles, se seleccionó a FeatureIDE y DeltaJ.

3.4.2.1. FeatureIDE

Se seleccionó a FeatureIDE porque integra diversas técnicas de implementación como Programación Orientada a Aspectos (AspectJ), Programación Orientada a Características y Programación Orientada a Deltas (DeltaJ) aunque solo funciona en versiones anteriores.

FeatureIDE apoya el desarrollo de LPS dentro de las siguientes 4 fases:

1. **Análisis de Dominio:** el resultado de este análisis se cubre en un modelo de características, el cual describe la variabilidad y los puntos comunes de un sistema de software basado en un dominio dado.

2. **Implementación de Dominio:** la implementación de la LPS descrita se lleva a cabo para todo el código base. Cada característica se mapea a artefactos de código.

3. **Análisis de requerimientos:** la adaptación de un sistema de software a las necesidades del cliente se realiza basándose en el dominio y las características. Los requisitos son mapeados manualmente a características, que se seleccionan según sea necesario para formar una configuración de un producto concreto.

4. **Generación de software:** FeatureIDE construye una variación seleccionada concreta automáticamente. La implementación del dominio y la generación de software dependen altamente unos de otros.

La Figura 3.2 muestra la representación gráfica de un modelo de características utilizando la herramienta FeatureIDE.

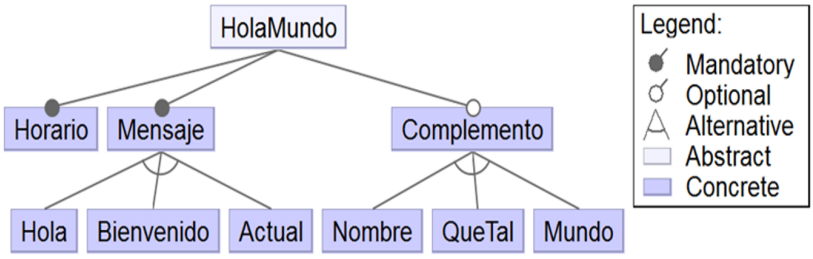


Figura 3.2: Definición de LPS en FeatureIDE

Como opción adicional, FeatureIDE ofrece la capacidad de exportar el diagrama de caracteristícas a otras notaciones: Velvet y Guidsl (Figura 3.3).

Velvet

```
concept HolaMundo {
        mandatory feature Horario;
        mandatory feature Mensaje {
                oneOf {
                feature Hola;
                feature Bienvenido;
                feature Actual;
                }
        }
        feature Complemento {
                oneOf {
                        feature Nombre;
                        feature QueTal;
                        feature Mundo;
                }
        }
}
```

Guidsl

```
HolaMundo: Horario Mensaje
[Complemento] :: _HolaMundo;

Mensaje: Hola
| Bienvenido
| Actual;

Complemento: Nombre
| QueTal
| Mundo;
```

Figura 3.3: Definición de LPS en otras notaciones

FeatureIDE permite configurar productos a partir de la selección de características del modelo de características. Sin embargo, solo es posible la configuración parcial de un producto y el ajuste a las restricciones de la LPS es de forma manual (Figura 3.4). Esto se convierte en algo tedioso por el número de combinaciones posibles.

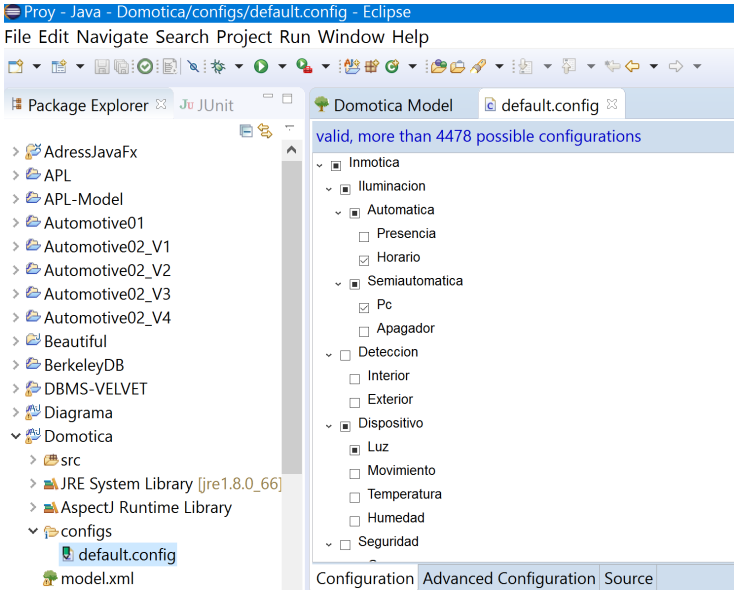


Figura 3.4: Configuración de un producto en FeatureIDE

Como opción adicional, FeatureIDE tiene disponibles algunas operaciones para el análisis de los modelos de características: calcula errores y redundancias en las restricciones, características muertas y número de posibles productos.

3.4.2.2. DeltaJ

DeltaJ es un lenguaje de programación basado en Java que agrega una capa de abstracción y permite organizar las clases en módulos. Hay dos clases de módulos: núcleo y delta. Un módulo núcleo es una simple colección de clases mientras un módulo delta es un conjunto de operaciones que permiten agregar, eliminar o modificar clases declaradas en otros módulos núcleo o delta.

DeltaJ trabaja a nivel de código fuente y permite definir las funcionalidades a implementar en la LPS, las combinaciones de estas funcionalidades definen un producto de software y los módulos deltas contienen las condiciones y modificaciones a realizar para cada producto.

Los proyectos DeltaJ incluyen tres carpetas de código fuente:

- **src:** esta es la carpeta del código fuente predeterminado. Normalmente contiene los módulos delta, los cuales se almacenan como archivos.
- **.deltaj.spl-info:** esta carpeta contiene el archivo de la declaración de la Línea de Productos de Software (SPLD). En este archivo se definen: características, módulos de deltas, restricciones que estructuran las características, se declaran las dependencias, el mapeo las n-m características y definición de productos.
- **src-gen:** esta carpeta contiene el código fuente generado para los productos. En el principio está vacío. El proceso de generación generará archivos fuente Java que luego se almacenarán en esta carpeta. Cada producto tiene su propia raíz de paquetes, que permite generar diferentes productos.

Cada proyecto DeltaJ se basa en proyectos de Java por defecto. Eso da la posibilidad de añadir los archivos fuente de Java en la carpeta src o unir las API externas que se usan dentro del código fuente de las LPS. La Figura 3.5 muestra la representación textual del modelo de características para declarar una Línea de Productos de Software en DeltaJ 1.5.

```
SPL Saludos {
Features = {Saludo, Horario, Mensaje, Hola, Bienvenido, Actual, Complemento, Nombre, QueTal, Mundo}
Deltas = {dBase, dHorario, dMensaje, dHola, dBienvenido, dActual, dComplemento, dNombre, dQueTal, dMundo}
Constraints {
        Saludo & Horario & Mensaje;
        Mensaje & (Hola ^ Bienvenido ^ Actual);
        Complemento & (Nombre ^ QueTal ^ Mundo);
}
Partitions {
        {dBase, dHorario, dMensaje} when (Horario);
        {dHola} when (Hola);

        {dBienvenido} when (Bienvenido);
        {dActual} when (Actual);
        {dNombre} when (Nombre);
        {dQueTal} when (QueTal);
        {dMundo} when (Mundo);
        {dComplemento, dNombre, dQueTal, dMundo} when (Complemento);
        }
Products {
        p1= {Horario, Hola};
        p2= {Horario, Hola, Nombre};
}
}
```

Figura 3.5: Definición de LPS en DeltaJ

Se seleccionó a DeltaJ porque ofrece una gran flexibilidad al configurar, definir y generar una serie de productos a partir de un conjunto de características y módulos delta que definen las reglas de combinación de los códigos fuente desarrollados. Asimismo, una de las LPS que se tiene como insumo se encuentra implementada en DeltaJ. Sin embargo, se identificó que al igual que FeatureIDE, es necesario cubrir actividades de la Ingeniería de Aplicación.

Se detectaron algunas desventajas al utilizar DeltaJ: (1) la sintaxis es compatible solo con la versión 1.5 de Java; (2) al editar un delta, la opción de autocompletar código no está disponible, lo que a veces resulta incómodo; y (3) los errores de semántica se detectan hasta generar los

productos.

A diferencia de FeatureIDE, DeltaJ genera un configurador que permite elegir el producto a desarrollar de los previamente declarados. Sin embargo, no permite configurar los deltas (seleccionar las características) en tiempo de ejecución.

3.5. Definición de las estrategias de composición

Generalmente, una MPL se integra de diversas LPS o varias partes de LPS con el fin de reducir la complejidad de los modelos de características y formar así un sistema de software. A continuación,se describen las 3 posibles opciones de integrar (composición) las LPS para derivar nuevos productos de software.

De acuerdo a la complejidad estructural [34, 140], es posible agrupar a las LPS a gran escala o MPL como:

- **Programa de Líneas de Productos (*Program of Product Lines*)**: este enfoque se emplea cuando se requiere extender el alcance de la LPS en términos de características. En este enfoque, se define una arquitectura de software para todo el sistema y se especifican los componentes que integran el sistema. Los componentes son algunas o todas las Líneas de Productos de Software desarrolladas por separado. Esta noción está relacionada con el Sistema de Sistemas. Por ejemplo, Nokia Networks desarrolla el software para sus conmutadores de telecomunicaciones como un programa de líneas de productos.

- **Línea de Productos jerárquica (*Hierarchical Product Line*):** este tipo consta de varias capas apiladas de Líneas de Productos. La base de la Línea de Productos comprende la funcionalidad compartida por todos los productos; mientras que las LPS de niveles inferiores especializan los niveles superiores. Las LPS jerárquicas son aplicables a situaciones donde el número y variabilidad de los productos es grande o muy grande y el número de personal involucrado en la producción de productos es grande. Esta noción está relacionada con el término de LPS anidadas [7]. Por ejemplo, Philips construye jerárquicamente

sus Líneas de Productos de dispositivos de imágenes médicas (resonancia magnética, radiografía y ultrasonido).

- **Población de productos:** cuando las organizaciones deciden aumentar el alcance en términos del número de productos, generalmente utilizan un enfoque de población del producto. Esto se refiere a la reutilización de la funcionalidad en varios dominios. Una población de productos es un conjunto de Líneas de Productos de Software que comparten componentes de un repositorio común de componentes. Cada LPS tiene su arquitectura dedicada, de la que derivan los productos. Este enfoque orientado a la composición es adecuado cuando el dominio que se necesita cubrir es muy amplio, de modo que el potencial de reutilización en general es pequeño, pero la reutilización a través de los subdominios es flexible. Esta noción está relacionada con las **Líneas de Productos Orientadas a Servicios *(SOPL, Service Oriented Product Line*)** [109], las cuales se refieren a una LPS que consumen productos suminitradas por LPS de terceros. La diferencia es que las poblaciones de productos se centran en las interfaces arquitectónicas. Por ejemplo, el *Koala component model*, permite la construcción de poblaciones de productos en el dominio de la electrónica de consumo.

Considerando la presencia de jerarquías entre las LPS [14], es posible agrupar las MPL como:

- **MPL Jerárquica:** las dependencias entre las instancias de las LPS de una MPL a menudo resultan en una jerarquía de LPS de múltiples niveles. Por lo general, la variabilidad de una MPL se describe mediante el modelo de características del nivel superior de una MPL.

- **MPL Plana:** este tipo de MPL no presentan una jerarquía. Una MPL plana ocurre cuando las LPS se desarrollan por separado y requieren configurarse para lograr compatibilidad entre ellas o cuando múltiples instancias de la misma LPS se combinan en una MPL. Por ejemplo: una MPL de programas de comunicación como Arquitectura cliente-servidor donde el cliente y servidor se desarrollan como LPS distintas y tienen que configurarse para lograr compatibilidad.

Considerando la complejidad a nivel organizacional, las MPL permiten construir:

- **Ecosistema de software (SECO):** cuando una compañia decide abrir su plataforma de LPS con el fin de permitir a otras desarrollar extensiones, crea un ecosistema de software. Es decir, el ecosistema de software es una forma efectiva de construir grandes sistemas de software sobre plataformas de software mediante la unión de varios componentes desarrollados por varios actores tanto internos como externos. En este entorno, la Ingeniería de Software se expande fuera de sus límites tradicionales de compañías de software a grupos de compañías, personas privadas u otras entidades legales. En [25] se diferencia entre ecosistemas enriquecidos con variabilidad (*variability-rich ecosystem*) y ecosistemas no enriquecidos con variabilidad (*non-variability-rich ecosystems*). Un **ecosistema enriquecido con variabilidad** es un ecosistema en donde la variabilidad de los activos producidos por una organización impacta fuertemente los activos producidos por una organización diferente. Este ecosistema requiere que las organizaciones sean conscientes de la variabilidad introducida en una organización diferente [23].

- **Cadena de suministro de software**: enfatizan el hecho de que el software se desarrolla a partir de componentes externos, comerciales y altamente configurables en lugar de desarollarlos. La arquitectura, organización y la administración de las dependencias requieren considerar que la LPS se distribuye por diferentes instituciones con diferentes estrategias, conocimientos e intereses.

3.5.1. Esquema para el desarrollo de una MPL

Mediante el análisis de la literatura, se obtuvo un esquema preliminar de cómo combinar y reutilizar insumos de las LPS para generar nuevos productos de software como componentes, sistemas, Líneas de Productos de Software, Sistemas de Sistemas o Ecosistemas de Software (Figura 3.6).

En un entorno de MPL, el configurador se encarga de utilizar, reutilizar o reconfigurar los insumos de diversas LPS de acuerdo a las necesidades y se agrupan en tres formas diferentes

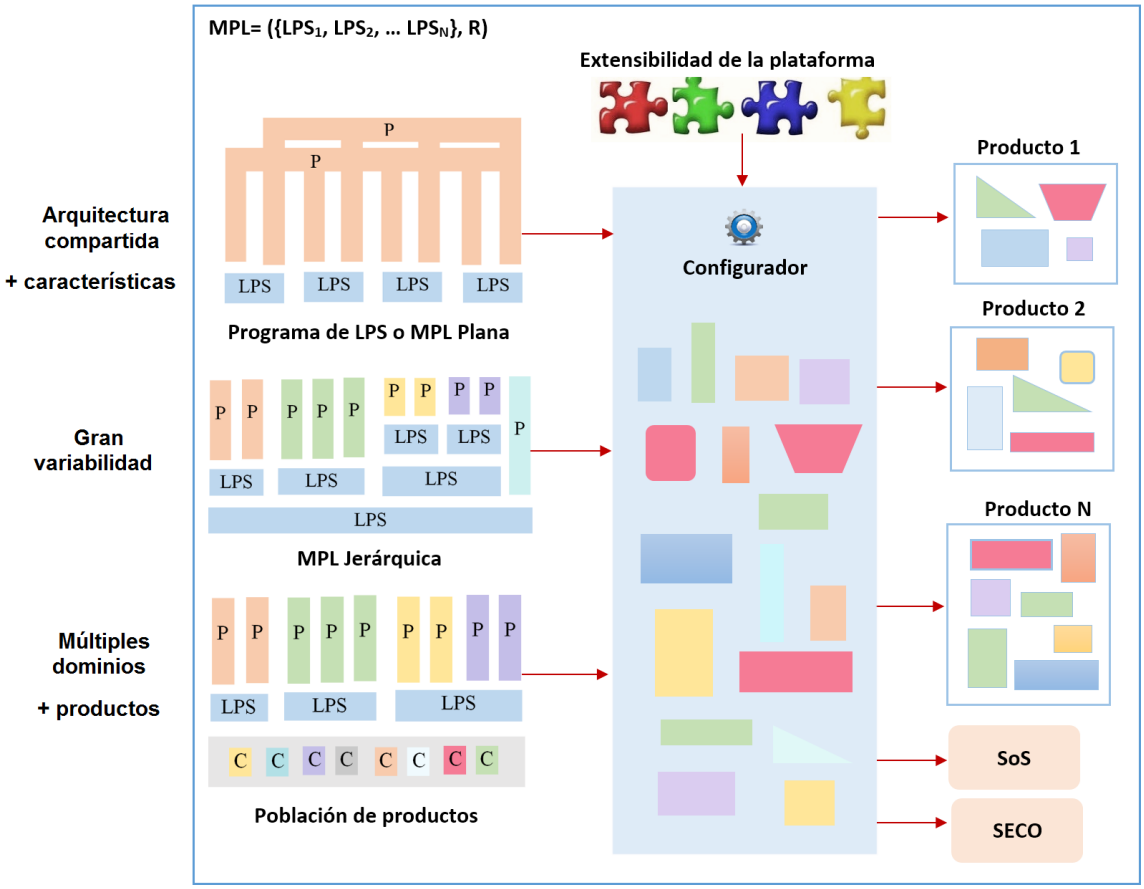


Figura 3.6: Esquema para el desarrollo de una MPL

con el fin de crear automáticamente instancias del producto. Sin embargo, los factores que determinan los límites de cada LPS dependen de la misión, la estructura, el mercado objetivo y la cultura de la organización que desarrolla el producto.

Mientras que una MPL Plana permite extender el alcance de la LPS en términos de características, una MPL Jerárquica es aplicable a casos en los que se tiene un gran numero de productos y variabilidad. Otra opción es la definición de una Población de productos que permite reutilizar la funcionalidad dispersa en diversos dominios.

3.6. Análisis de los enfoques para representar Arquitecturas de Referencia en las MPL

En esta sección se describe el único enfoque identificado para describir arquitecturas para MPL denominado **Archample (*Architectural Analysis Approach for Multiple Product Line Engineering***). A diferencia de los enfoques existentes de análisis de arquitecturas, Archample se enfoca en el análisis de la arquitectura de una MPL en particular (Aselsan REHIS).

El objetivo de Archample es apoyar la decisión sobre si es necesario utilizar una arquitectura MPL y asimismo evaluar diferentes alternativas de descomposición de la arquitectura MPL. Archample introduce puntos de vista arquitectónicos (*architectural viewpoints*) para el modelado y documentación de MPLs.

Archample involucra un grupo de stakeholders:

- **Decisores del proyecto:** gente interesada en el resultado de la evaluación y quienes afectan la dirección del proyecto. Estos tomadores de decisiones suelen ser administradores de proyectos.
- **Arquitecto MPL:** persona o equipo responsable del diseño de la arquitectura MPL y la coordinación del diseño de la sub-arquitectura.
- **Arquitecto LPS:** persona o equipo responsable del diseño de una sola arquitectura LPS. El único arquitecto LPS normalmente informa al arquitecto MPL sobre los resultados y, si es necesario, adapta la arquitectura.
- **Participantes en la arquitectura:** desarrolladores, testers, integradores, mantenedores, ingenieros de rendimiento, usuarios, constructores de sistemas, entre otros.
- **Evaluador (es) de arquitectura MPL:** persona o equipo responsable de la evaluación de la arquitectura MPL, así como la coordinación de la evaluación de las arquitecturas LPS.
- **Evaluador (es) de la arquitectura LPS:** persona o equipo responsable de la evaluación de la arquitectura LPS, así como de la coordinación de la evaluación de las arquitecturas MPL.

3.6.1. Proceso Archample

El enfoque de Archample consta de 4 fases: Preparación, Documentación del Diseño, Evaluación e Informes (Figura 3.7).

1. **Preparación**. Durante esta fase, primero se seleccionan las partes interesadas (stakeholders) y el equipo de evaluación (Paso 1). Después de seleccionar los stakeholdes, se planifica el cronograma de evaluación (Paso 2). En general, la evaluación completa de la MPL toma más tiempo que una evaluación de arquitectura única. Por lo tanto, para definir el cronograma se adopta un marco de tiempo más amplio que el habitual.

2. **Selección de la posible descomposición de MPL.** En esta fase, se proporcionan las diferentes alternativas de diseño (LPS, N LPS, AD LPS, CPL) de la arquitectura MPL (Paso 3), y se selecciona la alternativa factible (Paso 4). Las alternativas MPL se describen utilizando la descomposición de MPL y se utilizan puntos de vista arquitectónicos *(architectural viewpoints)*. La representación de la arquitectura MPL en el paso 3 es necesaria para garantizar que se proporcione la entrada adecuada al análisis en el paso 4. En esta etapa, no es necesario un diseño detallado de la MPL porque el diseño de una MPL es un proceso que consume mucho tiempo. Solo después de encontrar la descomposición factible en el paso 4, la documentación de diseño se completará en el paso 5.

3. **Evaluación de la alternativa de diseño de MPL seleccionada.** El paso 4 se enfoca en seleccionar una alternativa de descomposición de MPL factible. Una MPL consiste en varias LPS, por lo tanto, múltiples arquitecturas. Del mismo modo, en el paso 5 de Archample, es necesario centrarse en el análisis refinado de la alternativa MPL seleccionada. Es posible que la alternativa seleccionada sea una sola arquitectura de LPS o diferentes arquitecturas MPL. En caso de que la alternativa sea una CPL, se aplica un enfoque de evaluación por etapas en el que las unidades de MPL (PL o CPL) se evalúan recursivamente. Desde esta perspectiva, se distinguen dos tipos de evaluaciones: (a) evaluaciones de productos de arriba hacia abajo y (b) evaluaciones de productos de abajo hacia arriba.

4. **Informes y taller**. En la última fase de Archample, se proporciona un informe de los resultados de la evaluación y se organiza un taller con las partes interesadas.

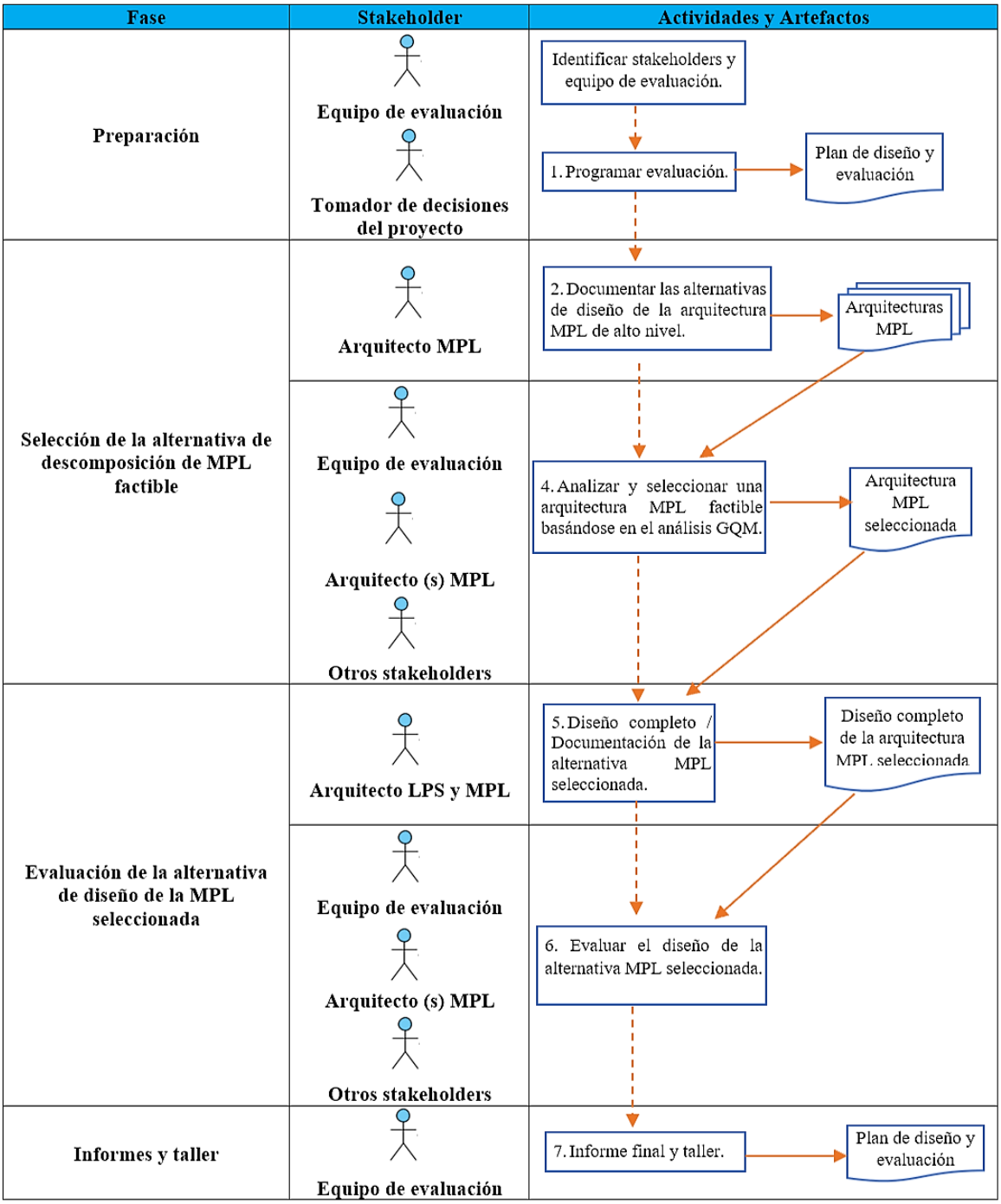


Figura 3.7: Proceso Archample

3.7. Definición del caso de estudio

Para validar el modelo matemático propuesto, se abordó el caso de estudio de una empresa privada en México que desarrolla y comercializa sistemas para el control y automatización inteligente de edificios principalmente hoteles a través de la generación de Líneas de Productos de Software (Domótica e Inmótica). Estos sistemas inmóticos permiten gestionar de forma eficiente el uso de energía, además de brindar seguridad, confort y comunicación entre el usuario y el sistema.

La **domótica** es el conjunto de tecnologías aplicadas al control y la automatización inteligente de la vivienda, que permite una gestión eficiente del uso de la energía, que aporta seguridad y confort, además de comunicación entre el usuario y el sistema. Un sistema domótico es capaz de recoger información proveniente de unos sensores o entradas, procesarla y emitir órdenes a unos actuadores o salidas.

La **inmótica** es el conjunto de tecnologías aplicadas al control y la automatización inteligente de edificios no destinados a vivienda, como hoteles, centros comerciales, escuelas, universidades, hospitales y todos los edificios terciarios, permitiendo una gestión eficiente del uso de la energía, además de aportar seguridad, confort, y comunicación entre el usuario y el sistema.

Básicamente, la diferencia radica en el tamaño del local, de esta forma una instalación domótica se refiere a aquella que se implanta en una vivienda unifamiliar mientras que la instalación Inmótica se refiere a aquella que se implanta en edificios de mayor envergadura como hospitales, hoteles, gimnasios, invernaderos, casa residencial, entre otros. Esto significa que los productos y servicios que nos facilita una instalación Inmótica son aplicables a una vivienda y a un edificio.

3.7.1. Problemática

La empresa de software decidió construir una MPL para rediseñar o reposicionar sus productos de software continuamente y generar un nuevo portafolio de productos que satisfaga diversos segmentos del mercado de automatización no sólo de interiores sino de exteriores como por ejemplo viviendas, oficinas, hoteles, jardines e invernaderos. La MPL debe reutilizar las dos líneas de productos de software previamente desarrolladas por diferentes equipos de desarrollo

y distinguidas principalmente por la tecnología empleada (lenguajes de programación, notaciones de modelado, hardware) y las cuales son capaces de generar productos de software para la automatización de una cadena de hoteles en México.

La problemática se centra en cómo desarrollar un configurador de múltiples líneas de productos de software para Inmótica (Building Automation System) que combine simultáneamente las tecnologías de software y hardware empleadas en la LPS1 (Home Automation System) con las tecnologías empleadas en la LPS2 (Building Automation System) y que sea capaz de obtener buenas configuraciones (combinaciones de características) con la suficiente compatibilidad y rapidez para integrarlo en diferentes espacios inteligentes.

Los ambientes inteligentes son entornos físicos equipados con tecnología que detectan y reaccionan a las actividades humanas y a los cambios en el entorno. Algunos ejemplos de espacios inteligentes son hogares, hoteles, hospitales, oficinas, granjas, invernaderos que reciben instrucciones y actúan de manera proactiva y/o reactiva a diferentes actividades.

De acuerdo a las LPS disponibles, las principales características que tienen la posibilidad de ser incluidas en los productos son las siguientes:

- Luces automáticas: la característica de iluminación está enfocada a la automatización de las habitaciones de las casas y edificios no dedicados a vivienda como hoteles, gimnasios, escuelas, entre otros. Esta característica busca satisfacer las expectativas de encender o apagar las luces de distintas formas (automática y semiautomática), para proporcionar un mayor confort a los habitantes y economizar el consumo de energía.

- Detector de incendios: la característica de detector de incendios está enfocada a ofrecer seguridad en las habitaciones de las instalaciones. Esta característica sirve para detectar a tiempo conatos de incendio y reducir o eliminar los riesgos generados por el fuego.

- Persianas automáticas: esta característica de persianas automáticas está enfocada a la automatización de las habitaciones o instalaciones. Esta característica permite satisfacer las expectativas de abrir o cerrar las persianas de una habitación y proporcionar un mayor confort a los usuarios.

- Energía inteligente: esta característica se enfoca a apoyar específicamente el ahorro de energía dentro de las habitaciones o instalaciones al mantener la luz encendida si hay alguna presencia humana o en determinados lapsos de tiempo. Esto beneficia la economía al reducir gastos y contribuye al desarrollo sustentable.

- Acondicionamiento de temperatura: esta característica se enfoca a satisfacer las expectativas de encender, apagar o controlar la potencia de aires acondicionados, calentadores, ventiladores o sistemas de calefacción para proporcionar un mayor confort a los habitantes.

3.7.2. Análisis de las LPS disponibles

Para el desarrollo de la MPL es de vital importancia la identificación de insumos disponibles. En esta actividad, se analizaron las LPS (Domótica e Inmótica) disponibles con el fin de identificar todos los aspectos y elementos que las describen principalmente en términos de proveedores, enfoques, tecnologías, características y productos.

Los insumos [1] disponibles para el desarrollo de la MPL son:

- Línea de Productos de Software para Domótica dirigida al dominio de domótica aplicando el enfoque composicional de DeltaJ y utilizando insumos de la empresa RODAS Computación S.A. de C.V.

- Línea de Productos de Software para el dominio de inmótica aplicando MDA (Arquitectura dirigida por modelos), Scala, AspectJ y utilizando insumos de la empresa RODAS Computación S.A. de C.V.

3.7.2.1. LPS Domótica

La figura 3.8 presenta el modelo de características de la LPS para Domótica utilizando como notación CVL.

[1] Las LPS de Domótica e Inmótica son el resultado de dos tesis de maestría, de las cuales se reutilizaron algunos artefactos como: requisitos, caracteristicas, modelos de características, hardware (sensores, Raspberry y Arduino), software, entre otros.

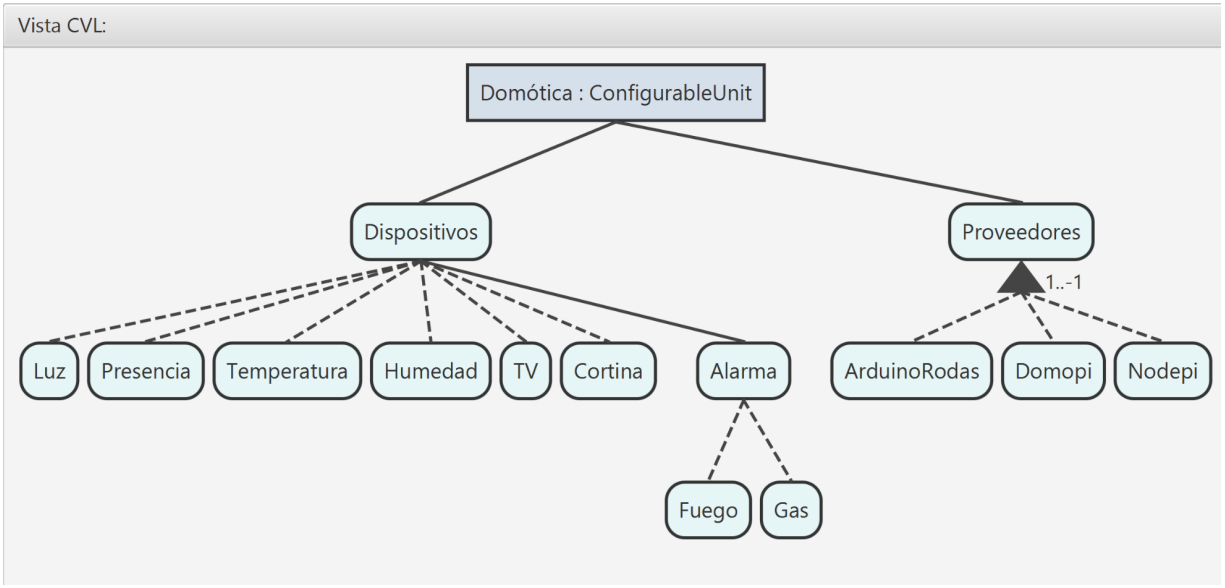


Figura 3.8: LPS Domótica

La Tabla 3.2 muestra las características conceptuales (features) que describen o distinguen los productos de la LPS para Domótica e integran la base para generar productos en la MPL. Las características (13) y productos (6) se implementaron utilizando DeltaJ. Por ejemplo, el producto plus se caracteriza por incluir el control de luces MiLight (dispositivos de control de focos RGB manufacturado por la empresa MiLight) y por agregar la funcionalidad de aceptar comandos en lenguaje natural (para aprovechar las API voz a texto que algunas plataformas ofrecen).

Tabla 3.2: Características y productos de la LPS de Domótica

Característica	Rodas	Domopi	Nodepi	Completo	Plus	Custom
Luces	✓	✓	✓	✓	✓	✓
Presencia	✓	✓	X	✓	✓	✓
Temperatura	✓	✓	✓	✓	✓	✓
Humedad	X	✓	✓	✓	✓	✓
TV	X	✓	X	✓	✓	✓
Cortinas	X	X	✓	✓	✓	✓
Alarma fuego	X	X	✓	✓	✓	✓

Continúa en la página siguiente

Tabla 3.2 Características y productos de la LPS de Domótica (continuación)

Característica	Rodas	Domopi	Nodepi	Completo	Plus	Custom
Alarma gases	X	X	✓	✓	✓	✓
Soporte Arduino Rodas	✓	X	X	✓	✓	X
Soporte Domopi (Raspberry)	X	✓	X	✓	✓	✓
Soporte Nodepi (NodeMCU)	X	X	✓	✓	✓	✓
Soporte de luces MiLight	X	X	X	X	✓	✓
Control por voz	X	X	X	X	✓	✓

Con base en lo revisado, el listado 3.1 describe todos los elementos necesarios para definir la LPS de Domótica en DeltaJ. Las líneas 3-7 indican la definición de características presentes en la LPS. Las líneas 9-15 definen los módulos de deltas e indican la creación de un archivo con extensión *.deltaj* por cada delta el cual incluye el código fuente.

Listado 3.1: intelidomo.spl

```
1  SPL intelidomo {
2
3      Features = {
4          base, postgresql, mysql, voz,
5          luces, presencia, temperatura, humedad, tv, cortina, alarma,
               fuego, gases,
6          rodas, domopi, nodemcu, milight
7      }
8
9      Deltas = {dBase, dDatabase, dModelos, dRules, dServer, dPgsql,
10         dLuces, dPresencia, dTemperatura, dHumedad, dTV, dCortina,
               dAlarma, dLuzRGB,
11         dRodas, dRodasLuces, dRodasPresencia, dRodasTemperatura,
12         dDomopi, dDomopiLuces, dDomopiPresencia, dDomopiTemperatura,
               dDomopiHumedad, dDomopiTV,
```

```
13          dNodepi, dNodepiLucesDim, dNodepiTemperatura, dNodepiHumedad,
                dNodepiCortina, dNodepiFuego, dNodepiGas,
14          dVoz, dVozLuces, dVozTemperatura, dVozHumd, dVozTV,
               dVozCortina,
15          dMilight
16      }
17
18      Constraints {
19          // Put boolean expression representation of the feature model
                here.
20          mysql ^ postgresql;
21          fuego & alarma;
22          gases & alarma;
23      }
24
25      Partitions {
26          {dBase,dDatabase,dModelos,dRules,dServer} when (base);
27          {dPgsql} when (postgresql);
28          {dLuces} when (luces);
29          {dPresencia} when (presencia);
30          {dTemperatura} when (temperatura);
31          {dHumedad} when (humedad);
32          {dCortina} when (cortina);
33          {dTV} when (tv);
34          {dAlarma} when (alarma);
35          {dLuzRGB} when (milight);
36          {dRodas} when (rodas);
37          {dRodasLuces} when (rodas & luces);
38          {dRodasPresencia} when (rodas & presencia);
39          {dRodasTemperatura} when (rodas & temperatura);
```

```
40          {dDomopi} when (domopi);
41          {dDomopiLuces} when (domopi & luces);
42          {dDomopiPresencia} when (domopi & presencia);
43          {dDomopiTemperatura} when (domopi & temperatura);
44          {dDomopiHumedad} when (domopi & humedad);
45          {dDomopiTV} when (domopi & tv);
46          {dNodepi} when (nodemcu);
47          {dNodepiLucesDim} when (nodemcu & luces);
48          {dNodepiTemperatura} when (nodemcu & temperatura);
49          {dNodepiHumedad} when (nodemcu & humedad);
50          {dNodepiCortina} when (nodemcu & cortina);
51          {dNodepiFuego} when (nodemcu & fuego);
52          {dNodepiGas} when (nodemcu & gases);
53          {dMilight} when (milight);
54          {dVoz} when (voz);
55          {dVozLuces} when (voz & luces);
56          {dVozHumd} when (voz & humedad);
57          {dVozTemperatura} when (voz & temperatura);
58          {dVozCortina} when (voz & cortina);
59          {dVozTV} when (voz & tv);
60      }
61
62      Products {
63          Rodas = {base, postgresql, luces, presencia, temperatura, rodas};
64          mio={base, postgresql, luces, presencia, temperatura, humedad,
                domopi};
65          Domopi = {base, postgresql, luces, presencia, temperatura, humedad
                , tv, domopi};
66          Nodepi = {base, postgresql, luces, temperatura, humedad, cortina,
                alarma, fuego, gases, nodemcu};
```

```
67          Completo = {base, postgresql, luces, presencia, temperatura,
                humedad, tv, cortina, alarma, fuego, gases, rodas, domopi, nodemcu
                };
68          Plus = {base, postgresql, luces, presencia, temperatura, humedad,
                tv, cortina, alarma, fuego, gases, rodas, domopi, nodemcu, milight
                , voz};
69          Personal = {base, postgresql, luces, presencia, temperatura,
                humedad, tv, cortina, alarma, fuego, gases, domopi, nodemcu, voz};
70      }
71  }
```

Las líneas 18-23 definen las restricciones del modelo de características. DeltaJ permite especificar restricciones alternativas, inclusivas o exclusivas para las características a través de operadores AND, OR y XOR. Asimismo, permite definir cuándo aplicar los deltas indicando una condición de características y el nombre del delta mediante la palabra reservada *when* (líneas 25-60). Finalmente, las líneas 62-69 indican la definición de productos.

3.7.2.2. LPS Inmótica

La figura 3.9 presenta el modelo de características de la LPS para Inmótica utilizando como notación CVL.

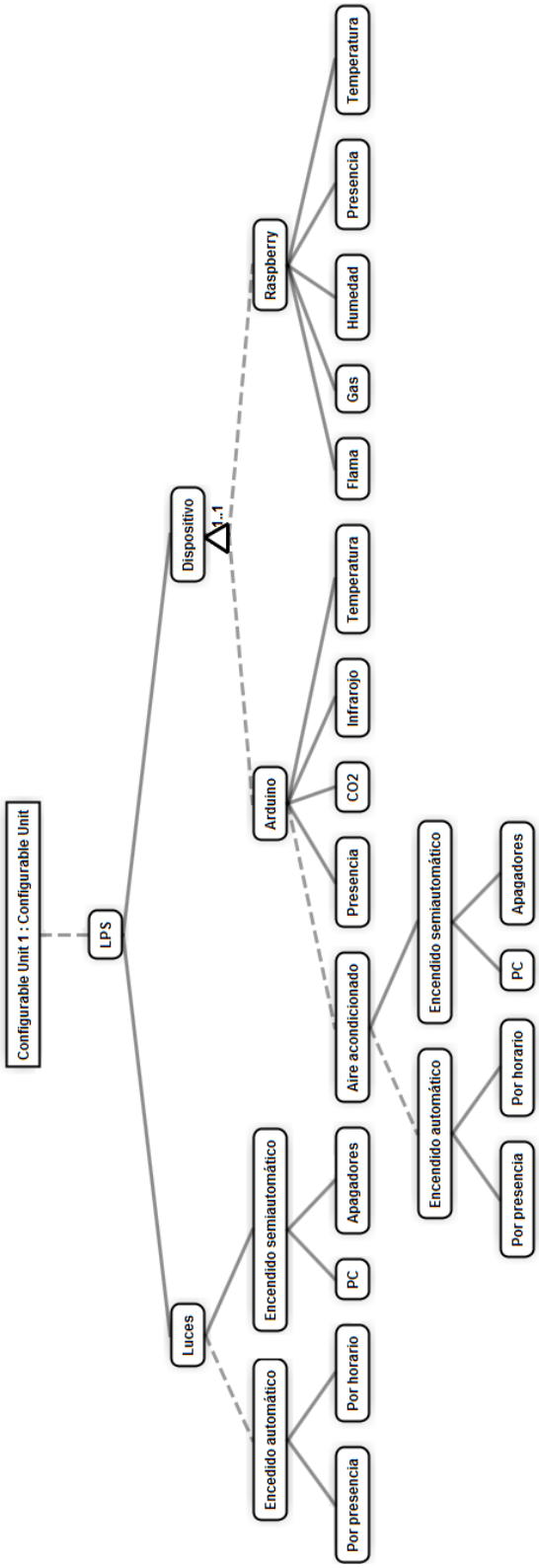


Figura 3.9: LPS Inmótica

La Tabla 3.3 muestra las características conceptuales (16) que describen o distinguen los productos (7) de la LPS para Inmótica e integran la base para generar productos en la MPL. Estas características y productos se implementaron mediante MDA, Scala y AspectJ.

Tabla 3.3: Características y productos de la LPS de Inmótica

Característica	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**
Control de luces por pc o laptop	✓	✓	✓	✓	✓	✓	✓
Automatización de luces con base en sensor presencia	✓	X	✓	X	✓	✓	X
Automatización de luces con base en horario	✓	X	✓	X	✓	✓	X
Dispositivo arduino	✓	✓	✓	✓	✓	X	X
Control de aire acondicionado por pc o laptop	✓	✓	X	X	✓	X	X
Automatización de aire acondicionado con base en presencia	✓	✓	X	X	X	X	X
Automatización de aire acondicionado con base en un horario	✓	✓	X	X	X	X	X
Incendios	✓	✓	✓	✓	✓	X	X
Sensor de CO2	✓	✓	✓	✓	✓	X	X
Sensor de presencia	✓	✓	✓	✓	✓	✓	✓
Sensor infrarrojo	✓	✓	✓	✓	✓	X	X
Sensor de temperatura	✓	✓	X	✓	✓	✓	✓
Dispositivo raspberry	X	X	X	X	X	✓	✓
Flama	X	X	X	X	X	✓	✓
Gas	X	X	X	X	X	✓	✓
Humedad	X	X	X	X	X	✓	✓

El funcionamiento del configurador de la LPS se basa principalmente en un archivo de configuración en formato XML. El contenido de este archivo se divide en 3 secciones principales: características, productos y recursos.

El listado 3.2 describe la definición del funcionamiento del configurador de la LPS Inmótica en un archivo XML. Las líneas 3-29 indican la definición de características presentes en la LPS (modelo de características). Las líneas 30-172 definen los productos (características que conforman una aplicación o producto) de la LPS, de tal forma que el configurador conoce cómo va a proceder para generar la aplicación que se le indique y las líneas 173-181 definen los recursos. Después de definir las secciones de características y productos se necesita conocer la ubicación de los diferentes artefactos que se obtienen como resultado de aplicar el *framework* de desarrollo. Los artefactos que se registran en el archivo de configuración son: componentes de software, requerimientos y diagramas de clases.

Listado 3.2: spl.xml

```
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <lps nombre="LPSok" ubicacion="C:/Users/gittz/Desktop/Aplicacions␣
       configurador">
3      <caracteristicas>
4          <caracteristica nombre="Luces"/>
5          <caracteristica nombre="Encendido␣luces␣automatico"/>
6          <caracteristica nombre="Luces␣por␣presencia"/>
7          <caracteristica nombre="Luces␣por␣horario"/>
8          <caracteristica nombre="Encendido␣luces␣semiautomaico"/>
9          <caracteristica nombre="Luces␣por␣latop␣o␣PC"/>
10         <caracteristica nombre="Apagadores␣de␣habitacion"/>
11         <caracteristica nombre="Arduino"/>
12         <caracteristica nombre="Raspberry"/>
13         <caracteristica nombre="Aire␣acondicionado"/>
14         <caracteristica nombre="Aire␣acondicionado␣encendido␣
               semiautomatico"/>
```

```
15          <caracteristica nombre="Aire␣acondicionado␣por␣laptop␣o␣PC"/>
16          <caracteristica nombre="Encendido␣aire␣acondicionado␣
                automatico"/>
17          <caracteristica nombre="Encendido␣aire␣acondicionado␣por␣
                presencia"/>
18          <caracteristica nombre="Encendido␣aire␣acondicionado␣por␣
                horario"/>
19          <caracteristica nombre="Incendio"/>
20          <caracteristica nombre="CO2"/>
21          <caracteristica nombre="Infrarojo"/>
22          <caracteristica nombre="Temperatura"/>
23          <caracteristica nombre="Flama"/>
24          <caracteristica nombre="Gas"/>
25          <caracteristica nombre="Humedad"/>
26          <caracteristica nombre="Obtener␣todos"/>
27          <caracteristica nombre="Test"/>
28          <caracteristica nombre="Cambiar␣temperatura"/>
29      </caracteristicas>
30      <productos>
31          <producto nombre="Producto␣1">
32              <caracteristica nombre="Base"/>
33              <caracteristica nombre="Luces"/>
34              <caracteristica nombre="Encendido␣luces␣automatico"/>
35              <caracteristica nombre="Luces␣por␣presencia"/>
36              <caracteristica nombre="Luces␣por␣horario"/>
37              <caracteristica nombre="Encendido␣luces␣semiautomaico"/>
38              <caracteristica nombre="Luces␣por␣latop␣o␣PC"/>
39              <caracteristica nombre="Apagadores␣de␣habitacion"/>
40              <caracteristica nombre="Arduino"/>
41              <caracteristica nombre="Aire␣acondicionado"/>
```

```
42              <caracteristica nombre="Aire␣acondicionado␣encendido␣
                    semiautomatico"/>
43              <caracteristica nombre="Aire␣acondicionado␣por␣laptop␣o␣
                    PC"/>
44              <caracteristica nombre="Encendido␣aire␣acondicionado␣
                    automatico"/>
45              <caracteristica nombre="Encendido␣aire␣acondicionado␣por␣
                    presencia"/>
46              <caracteristica nombre="Encendido␣aire␣acondicionado␣por␣
                    horario"/>
47              <caracteristica nombre="Incendio"/>
48              <caracteristica nombre="CO2"/>
49              <caracteristica nombre="Infrarojo"/>
50              <caracteristica nombre="Presencia"/>
51              <caracteristica nombre="Test"/>
52              <caracteristica nombre="Cambiar␣temperatura"/>
53          </producto>
54          <producto nombre="Producto␣2">
55              <caracteristica nombre="Base"/>
56              <caracteristica nombre="Luces"/>
57              <caracteristica nombre="Encendido␣luces␣semiautomaico"/>
58              <caracteristica nombre="Luces␣por␣latop␣o␣PC"/>
59              <caracteristica nombre="Apagadores␣de␣habitacion"/>
60              <caracteristica nombre="Arduino"/>
61              <caracteristica nombre="Aire␣acondicionado"/>
62              <caracteristica nombre="Aire␣acondicionado␣encendido␣
                    semiautomatico"/>
63              <caracteristica nombre="Aire␣acondicionado␣por␣laptop␣o␣
                    PC"/>
```

```
64              <caracteristica nombre="Encendido␣aire␣acondicionado␣
                    automatico"/>
65              <caracteristica nombre="Encendido␣aire␣acondicionado␣por␣
                    presencia"/>
66              <caracteristica nombre="Encendido␣aire␣acondicionado␣por␣
                    horario"/>
67              <caracteristica nombre="Incendio"/>
68              <caracteristica nombre="CO2"/>
69              <caracteristica nombre="Infrarojo"/>
70              <caracteristica nombre="Presencia"/>
71              <caracteristica nombre="Test"/>
72              <caracteristica nombre="Cambiar␣temperatura"/>
73          </producto>
74          <producto nombre="Producto␣3">
75              <caracteristica nombre="Base"/>
76              <caracteristica nombre="Luces"/>
77              <caracteristica nombre="Encendido␣luces␣automatico"/>
78              <caracteristica nombre="Luces␣por␣presencia"/>
79              <caracteristica nombre="Luces␣por␣horario"/>
80              <caracteristica nombre="Encendido␣luces␣semiautomaico"/>
81              <caracteristica nombre="Luces␣por␣latop␣o␣PC"/>
82              <caracteristica nombre="Apagadores␣de␣habitacion"/>
83              <caracteristica nombre="Arduino"/>
84              <caracteristica nombre="Incendio"/>
85              <caracteristica nombre="CO2"/>
86              <caracteristica nombre="Infrarojo"/>
87              <caracteristica nombre="Presencia"/>
88              <caracteristica nombre="Test"/>
89          </producto>
90          <producto nombre="Producto␣4">
```

```
91              <caracteristica nombre="Base"/>
92              <caracteristica nombre="Luces"/>
93              <caracteristica nombre="Encendido␣luces␣semiautomaico"/>
94              <caracteristica nombre="Luces␣por␣latop␣o␣PC"/>
95              <caracteristica nombre="Apagadores␣de␣habitacion"/>
96              <caracteristica nombre="Arduino"/>
97              <caracteristica nombre="Incendio"/>
98              <caracteristica nombre="CO2"/>
99              <caracteristica nombre="Infrarojo"/>
100             <caracteristica nombre="Presencia"/>
101             <caracteristica nombre="Test"/>
102             <caracteristica nombre="Cambiar␣temperatura"/>
103         </producto>
104         <producto nombre="Producto␣5">
105             <caracteristica nombre="Base"/>
106             <caracteristica nombre="Luces"/>
107             <caracteristica nombre="Encendido␣luces␣automatico"/>
108             <caracteristica nombre="Luces␣por␣presencia"/>
109             <caracteristica nombre="Luces␣por␣horario"/>
110             <caracteristica nombre="Encendido␣luces␣semiautomaico"/>
111             <caracteristica nombre="Luces␣por␣latop␣o␣PC"/>
112             <caracteristica nombre="Apagadores␣de␣habitacion"/>
113             <caracteristica nombre="Arduino"/>
114             <caracteristica nombre="Aire␣acondicionado"/>
115             <caracteristica nombre="Aire␣acondicionado␣encendido␣
                    semiautomatico"/>
116             <caracteristica nombre="Aire␣acondicionado␣por␣laptop␣o␣
                    PC"/>
117             <caracteristica nombre="Incendio"/>
118             <caracteristica nombre="CO2"/>
```

```
119             <caracteristica nombre="Infrarojo"/>
120             <caracteristica nombre="Presencia"/>
121             <caracteristica nombre="Test"/>
122             <caracteristica nombre="Cambiar temperatura"/>
123         </producto>
124         <producto nombre="Producto 6">
125             <caracteristica nombre="Base"/>
126             <caracteristica nombre="Luces"/>
127             <caracteristica nombre="Encendido luces semiautomaico"/>
128             <caracteristica nombre="Luces por latop o PC"/>
129             <caracteristica nombre="Apagadores de habitacion"/>
130             <caracteristica nombre="Arduino"/>
131             <caracteristica nombre="Aire acondicionado"/>
132             <caracteristica nombre="Aire acondicionado encendido
                    semiautomatico"/>
133             <caracteristica nombre="Aire acondicionado por laptop o
                    PC"/>
134             <caracteristica nombre="Incendio"/>
135             <caracteristica nombre="CO2"/>
136             <caracteristica nombre="Infrarojo"/>
137             <caracteristica nombre="Presencia"/>
138             <caracteristica nombre="Test"/>
139             <caracteristica nombre="Cambiar temperatura"/>
140         </producto>
141         <producto nombre="Producto 7">
142             <caracteristica nombre="Base"/>
143             <caracteristica nombre="Luces"/>
144             <caracteristica nombre="Encendido luces automatico"/>
145             <caracteristica nombre="Luces por presencia"/>
146             <caracteristica nombre="Luces por horario"/>
```

```
147             <caracteristica nombre="Encendido luces semiautomaico"/>
148             <caracteristica nombre="Luces por latop o PC"/>
149             <caracteristica nombre="Apagadores de habitacion"/>
150             <caracteristica nombre="Raspberry"/>
151             <caracteristica nombre="Temperatura"/>
152             <caracteristica nombre="Flama"/>
153             <caracteristica nombre="Gas"/>
154             <caracteristica nombre="Humedad"/>
155             <caracteristica nombre="Presencia"/>
156             <caracteristica nombre="Obtener todos"/>
157         </producto>
158         <producto nombre="Producto 8">
159             <caracteristica nombre="Base"/>
160             <caracteristica nombre="Luces"/>
161             <caracteristica nombre="Encendido luces semiautomaico"/>
162             <caracteristica nombre="Luces por latop o PC"/>
163             <caracteristica nombre="Apagadores de habitacion"/>
164             <caracteristica nombre="Raspberry"/>
165             <caracteristica nombre="Temperatura"/>
166             <caracteristica nombre="Flama"/>
167             <caracteristica nombre="Gas"/>
168             <caracteristica nombre="Humedad"/>
169             <caracteristica nombre="Presencia"/>
170             <caracteristica nombre="Obtener todos"/>
171         </producto>
172     </productos>
173     <recursos>
174         <recurso id_recurso="1" tipo="Requerimientos" url="C:/Users/
                gittz/Documents/Recursos/Requerimientos.xml"/>
```

```
175         <recurso id_recurso="2" tipo="Archivos_configuracion" url="C
                :/Users/gittz/Documents/Recursos/Archivos_configuracion"/>
176         <recurso id_recurso="3" tipo="Cliente" url="C:/Users/gittz/
                Desktop/ClienteLPS"/>
177         <recurso id_recurso="4" tipo="Servidor" url="C:/Users/gittz/
                Desktop/ServerLPS"/>
178         <recurso id_recurso="5" tipo="Clases_cliente_xml" url="C:/
                Users/gittz/Documents/Recursos/cliente.xml"/>
179         <recurso id_recurso="6" tipo="Clases_servidor_xml" url="C:/
                Users/gittz/Documents/Recursos/servidor.xml"/>
180         <recurso id_recurso="7" tipo="Componentes_xml" url="C:/Users/
                gittz/Documents/Recursos/componentes.xml"/>
181     </recursos>
182 </lps>
```

3.8. Diseño y solución de un modelo matemático

En un modelo matemático de optimización existe un conjunto de variables de decisión que deben maximizar/minimizar una función objetivo sometidas a un conjunto de restricciones.

Los elementos que integran el modelo matemático de optimización son:

- **Función objetivo:** medida cuantitativa del funcionamiento del sistema que se desea maximizar o minimizar.
- **Variables:** representan las decisiones que es posible tomar para afectar el valor de la función objetivo.
- **Restricciones:** representan el conjunto de relaciones (ecuaciones o inecuaciones) que ciertas variables están obligadas a satisfacer.

Las etapas en el desarrollo de un modelo matemático de optimización son:

1. Identificación del problema: esta etapa consiste en recolectar y analizar la información relevante para el problema.

2. Formulación matemática: esta etapa se refiere a la especificación matemática del problema de optimización en la cual se definen las variables, ecuaciones, funciones objetivo y parámetros.

3. Resolución: en esta etapa se resuelve el problema de optimización y consiste en encontrar el valor que deben tomar las variables para obtener un valor óptimo en la función objetivo satisfaciendo el conjunto de restricciones. Los métodos de optimización se clasifican en clásicos y metaheurísticos. Los métodos clásicos buscan y garantizan un óptimo local mientras que los metaheurísticos tienen mecanismo para alcanzar un óptimo global.

4. Verificación y validación del modelo: esta etapa se refiere a comprobar la validez del modelo.

5. Interpretación y análisis de los resultados: esta etapa consiste en proponer soluciones. Permite conocer el comportamiento del modelo con los parámetros de entrada, estudiar diferentes escenarios, detectar soluciones alternativas y óptimas.

3.9. Desarrollo de los artefactos necesarios para la implementación de la MPL

En esta sección se describen los artefactos desarrollados para apoyar el desarrollo de la MPL.

3.9.1. Administración de la variabilidad de una MPL

MPL necesita administrar las LPS de manera eficiente para identificar los puntos de variación y el nivel de reutilización, y así configurar nuevos productos en diferentes dominios.

Para generar productos en una MPL y reutilizar artefactos de las LPS disponibles es recomendable fusionar sus modelos de características y determinar el alcance que tendrá una MPL. El alcance de una MPL es una descripción de los productos que constituyen la MPL o la capacidad de producción de una MPL de *N* líneas de productos de software. Dentro de ese ámbito, la reutilización disciplinada de los activos principales, como los requisitos, características, diseños, casos de prueba, herramientas y otros artefactos de desarrollo de software, reduce en gran medida el costo del desarrollo de productos.

Las actividades identificadas para administrar la variabilidad de una MPL son:

- Identificar el punto de variación de cada modelo de características de las LPS y las características comunes de todas las LPS.
- Comparar las características comunes y variables del dominio y encontrar los puntos de variación comunes de MPL.
- Mapear los puntos de variación comunes en el repositorio único para el desarrollo inicial y la reutilización en desarrollos posteriores del producto.

3.9.1.1. Fusión de modelos de características

Cuando dos o más modelos de características comparten varias características, es necesario combinar/fusionar las partes superpuestas y obtener un único modelo que presente una visión integrada del sistema. A este proceso se le denomina **fusión de modelos de características**.

El objetivo de la fusión de modelos es tradicionalmente agrupar (combinar) elementos de modelo que describen los mismos conceptos en los modelos de entrada que se van a componer [141]. En el contexto de los modelos de características, se considera que los elementos del modelo son las características de las LPS y que dos características coinciden (es decir, representan los mismos conceptos) si tienen el mismo nombre.

El conjunto de características del modelo resultante de combinar múltiples modelos de características es el super-conjunto de todas las características de los modelos. Sin embargo, considerando que los modelos de características representan el conjunto de productos (configura-

ciones) en una línea de productos, es posible que la semántica del modelo resultante varíe de una aproximación a otra.

En la literatura, se reportan 3 tipos de operaciones para fusionar modelos de características [142]:

- **Intersección:** cuando la fusión resulta en un modelo de características que representa solo los productos que existen en los modelos constituyentes. La operación de fusión-intersección toma dos modelos de características FM_1 y FM_2 y produce un modelo de características resultante FM_r (Figura 3.10) tal que el conjunto de configuraciones que son válidas resulta de la intersección de las configuraciones válidas de los modelos fuente, es decir, $[FM_r] = [FM_1] \cap [FM_2]$.

  - $[FM_1] = \{\{r,A,B,C\},\{r,A,B,D\},\{r,B,C\},\{r,B,D\}\}$
  - $[FM_2] = \{\{r,A\},\{r,A,B,C\},\{r,A,B,E\}\}$
  - $[FM_r] = \{\{r,A,B,C\}\}$

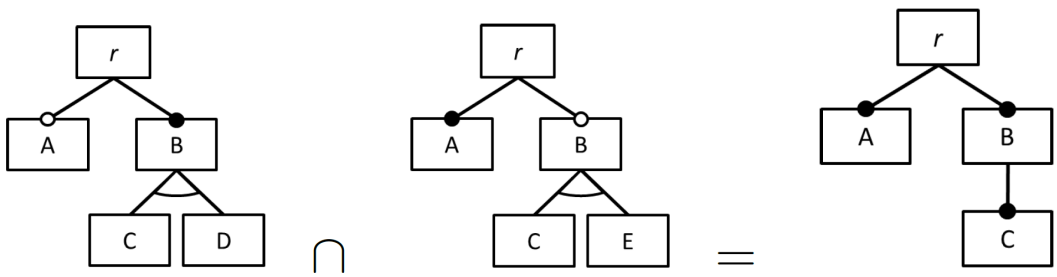


Figura 3.10: Ejemplo de fusión-intersección de FMs

Es posible que de la fusión por intersección resulte un modelo de características vacío cuando la intersección de las configuraciones válidas de los modelos para fusionar está vacía. Por ejemplo, considere los modelos de características representados en la Figura 3.11. Considerando que la intersección de las configuraciones de estos modelos está vacía, es decir, $[FM_r] = [FM_1] \cap [FM_2] = 0$, la intersección falla.

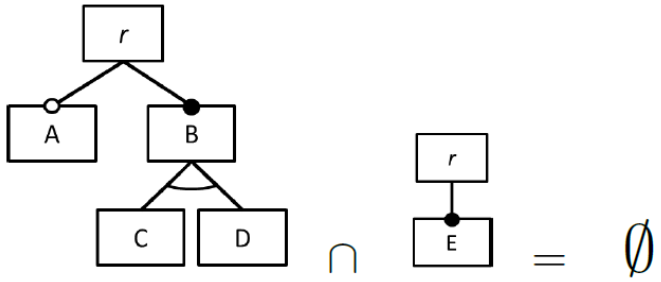


Figura 3.11: Ejemplo de fusión-intersección cuando produce un error

- **Unión:** cuando la fusión resulta en un modelo de características que representa la unión de los productos de todos los modelos. La operación de fusión-unión (estricta) toma dos modelos de características FM_1 y FM_2 y produce un modelo de características resultante FM_r (Figura 3.12) con la unión de todas las configuraciones válidas de los modelos fuente, es decir, $[FM_r] = [FM_1] \cup [FM_2]$.

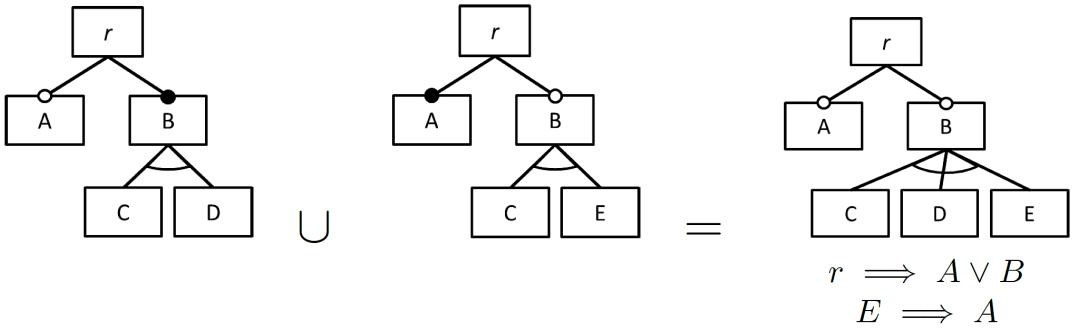


Figura 3.12: Ejemplo de fusión-unión de FMs

  - $[FM_r] = \{\{r,A\},\{r,A,B,C\},\{r,A,B,D\},\{r,A,B,E\},\{r,B,C\},\{r,B,D\}\}$

- **Producto reducido:** cuando el modelo de características resultante incluye productos que combinan las características de cada producto válido de un modelo con las características de los productos válidos de otro modelo. La agregación reducida de producto toma dos modelos de características FM_1 y FM_2 y produce un modelo de característica FM_r tal que las configuraciones que son válidas son el producto reducido de las configuraciones válidas de los modelos fuente, es decir:

- $[FM_r] = [FM_1] \otimes [FM_2] = C_1 \cup C_2 | C_1 \in [FM_1] \wedge C_2 \in [FM_2]$

Por ejemplo, considerando los mismos modelos presentados anteriormente. El producto reducido (o agregación) producirá un modelo de características donde el conjunto de configuraciones válidas es la combinación de las configuraciones válidas de ambos modelos. La figura 3.13 muestra los resultados de la agregación.

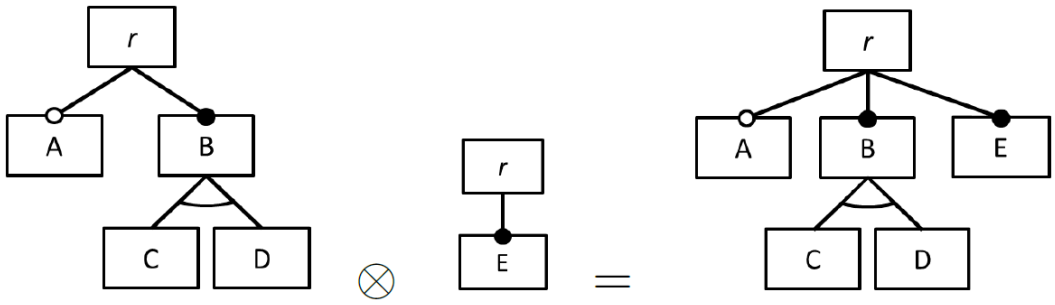


Figura 3.13: Ejemplo de agregación

- $[FM_r] = \{\{r,A,B,C,E\},\{r,A,B,D,E\},\{r,B,C,E\},\{r,B,D,E\}\}$

A diferencia de las otras operaciones de combinación, es posible utilizar la agregación de productos reducida para combinar modelos de características ortogonales (o casi ortogonales) que representan diferentes aspectos o dominios técnicos de los mismos productos.

3.9.1.2. Matriz de requisitos de aplicaciones de la MPL

Para apoyar el desarrollo de la MPL Inmótica, es necesario construir el artefacto denominado matriz de requisitos de aplicaciones de la MPL. La matriz de requisitos de aplicaciones permite identificar que requisitos son comunes y variables para todas las aplicaciones de las LPS que integran la MPL (Tabla 3.4).

- Las filas representan los requisitos de las aplicaciones y las columnas representan los productos o aplicaciones de cada LPS (Domótica e Inmótica.)

- Las celdas (✓) identifican los requisitos presentes en cada producto.

- Las celdas vacías indican que el producto no incluye ese requisito. De esta forma los requisitos que no aparecen en alguna aplicación son candidatos a ser requisitos variables.

Tabla 3.4: Matriz de requisitos de la MPL

Requisito	**Productos Domótica**							**Productos Inmótica**					
	P1	P2	P3	P4	P5	P6	P7	P1	P2	P3	P4	P5	P6
Autenticarse para iniciar la aplicación.	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Pantalla de inicio de la aplicación.	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Visualizar las peticiones que se realizan al servidor.	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Opción para detener el servidor cuando se requiera.	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Control de luces por pc o laptop.	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Automatización de luces con base en sensor de presencia.	✓		✓		✓	✓							
Automatización de luces con base en horario.	✓		✓		✓	✓							
Control de aire acondicionado por pc o laptop.	✓	✓			✓								
Automatización de aire acondicionado con base en presencia.	✓	✓											
Automatización de aire acondicionado con base en un horario.	✓	✓											
Control de incendios / fuego.	✓	✓	✓	✓	✓						✓	✓	✓
Cada área tendrá un sensor de detección de CO2.	✓	✓	✓	✓	✓								
Cada área tendrá instalado un sensor de presencia.	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓
Cada área tendrá un sensor infrarrojo.	✓	✓	✓	✓	✓								
Cada área tendrá un sensor de temperatura.	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Utilizar dispositivo Arduino para la aplicación.	✓	✓	✓	✓	✓			✓			✓	✓	
Utilizar dispositivo Raspberry para la aplicación.						✓	✓		✓		✓	✓	✓
Cada área tendrá sensor de flama y se visualizará su estado en pantalla.						✓	✓						
Cada área tendrá sensor de gas y se visualizará su estado en pantalla.						✓	✓						
Cada área tendrá instalado un sensor de humedad.						✓	✓		✓	✓	✓	✓	✓
Control de TV.									✓		✓	✓	✓
Control de cortinas / persianas.										✓	✓	✓	✓
Alarma gases											✓	✓	✓
Control por voz												✓	✓
Visualizar el estado que registran todos los sensores	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

3.9.1.3. Modelo de características de la MPL

Otro artefacto necesario para el desarrollo de la MPL es el modelo de características universal de las LPS denominado Modelo de características MPL (FM_{MPL}). Mediante el análisis de las LPS, se elaboró el modelo de características de la MPL Inmótica (Figura 3.14) que representa todas las características de las LPS y sus interrelaciones. La característica fundamental es la MPL y sus sub-características son los modelos de características de las familias de productos en las MPLs. Cabe resaltar que el modelo de características se elaboró con los resultados obtenidos por la optimización del modelo matemático utilizando la notación gráfica de la herramienta FeatureIDE para describir las características comunes y variables de las LPS.

La figura 3.15 muestra el análisis automático del modelo de características de la MPL utilizando a FeatureIDE como herramienta.

3.9.2. Arquitectura MPL

La arquitectura MPL representa una estructura a un gran nivel del sistema que consiste en subproductos derivados de LPS separadas que en conjunto forman un producto. Una arquitectura MPL se considera una arquitectura de SoS que define las decisiones de diseño. Para la definición de la Arquitectura de Referencia de la MPL (Figura 3.18) que represente los activos comunes y variables de las LPS que integran la MPL, en primer lugar se modeló la arquitectura de software de la LPS Domótica (Figura 3.16) la cual no se tenía disponible y se redefinió la LPS Inmótica (Figura 3.17).

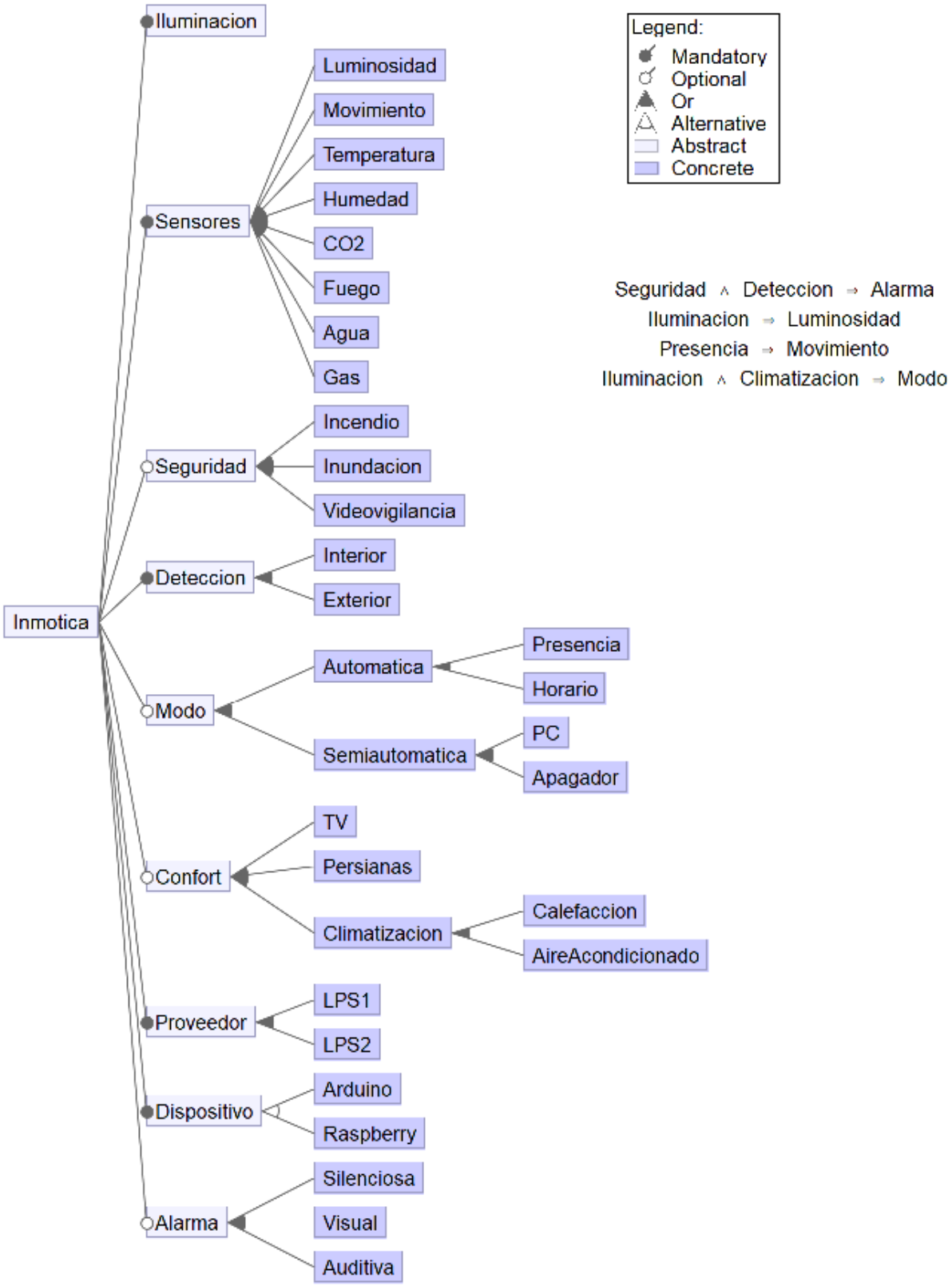


Figura 3.14: Modelo de características de la MPL Inmótica

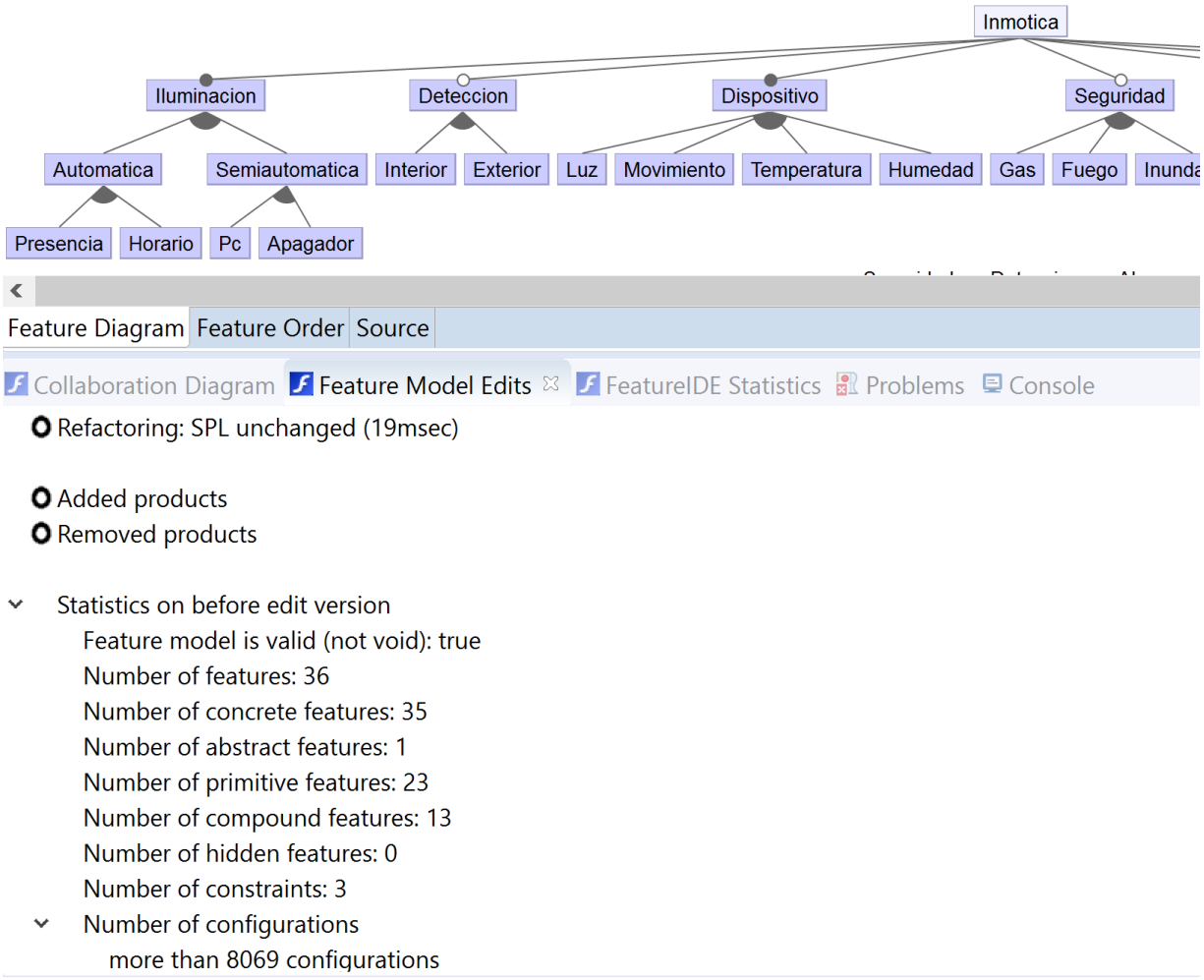


Figura 3.15: Análisis del modelo de características parcial de la MPL Inmótica

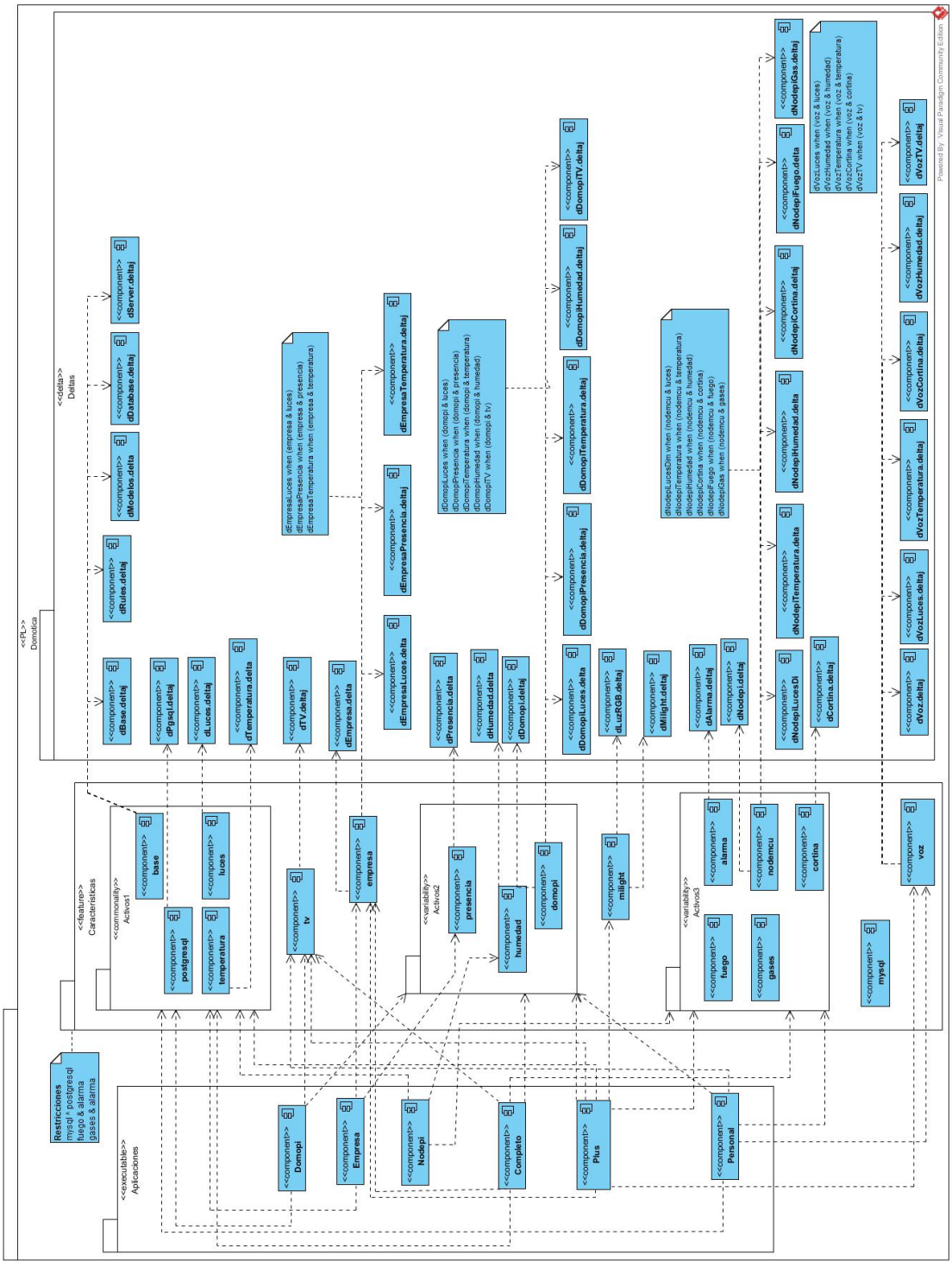


Figura 3.16: Arquitectura de la LPS Domótica

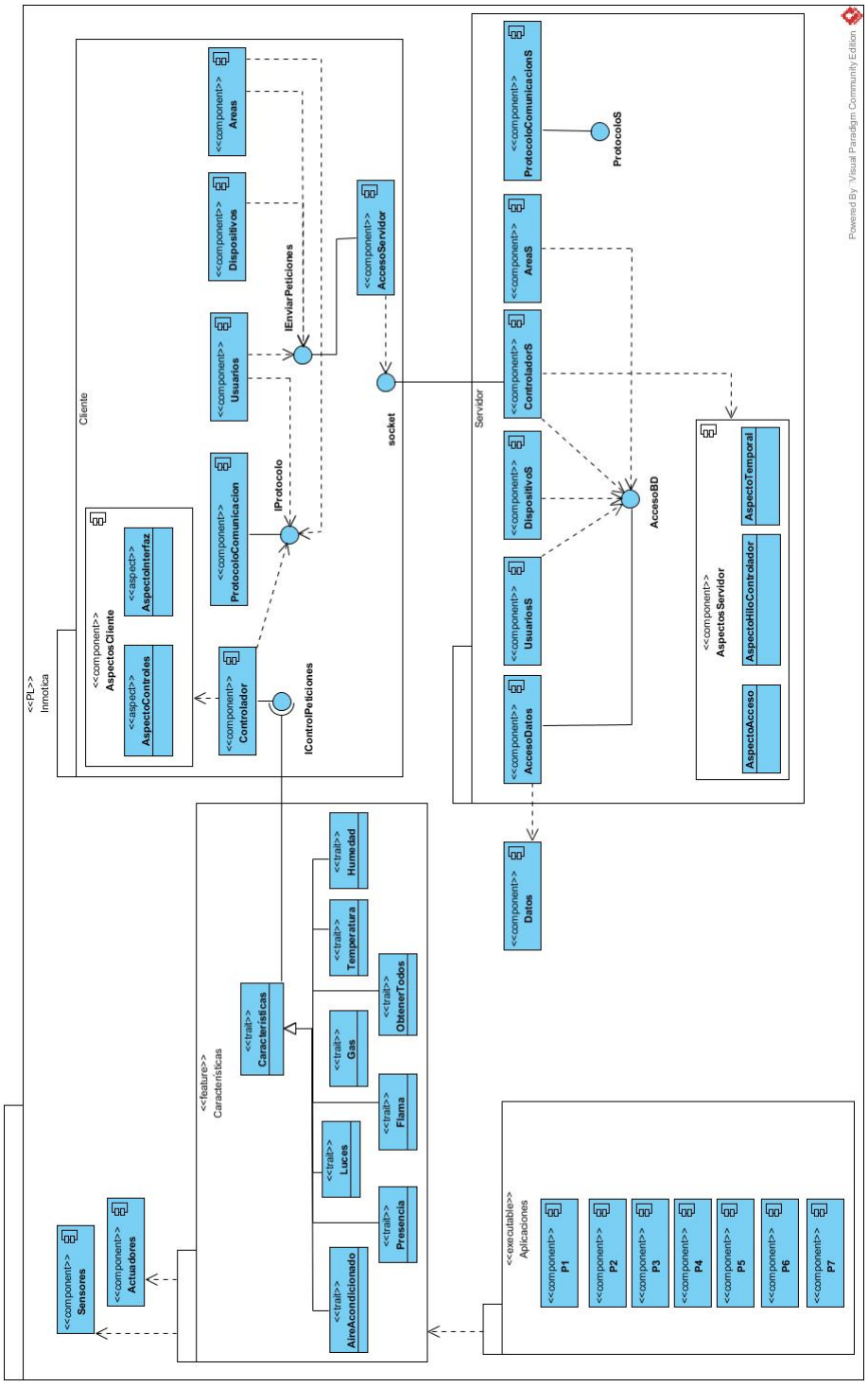


Figura 3.17: Arquitectura de la LPS Inmótica

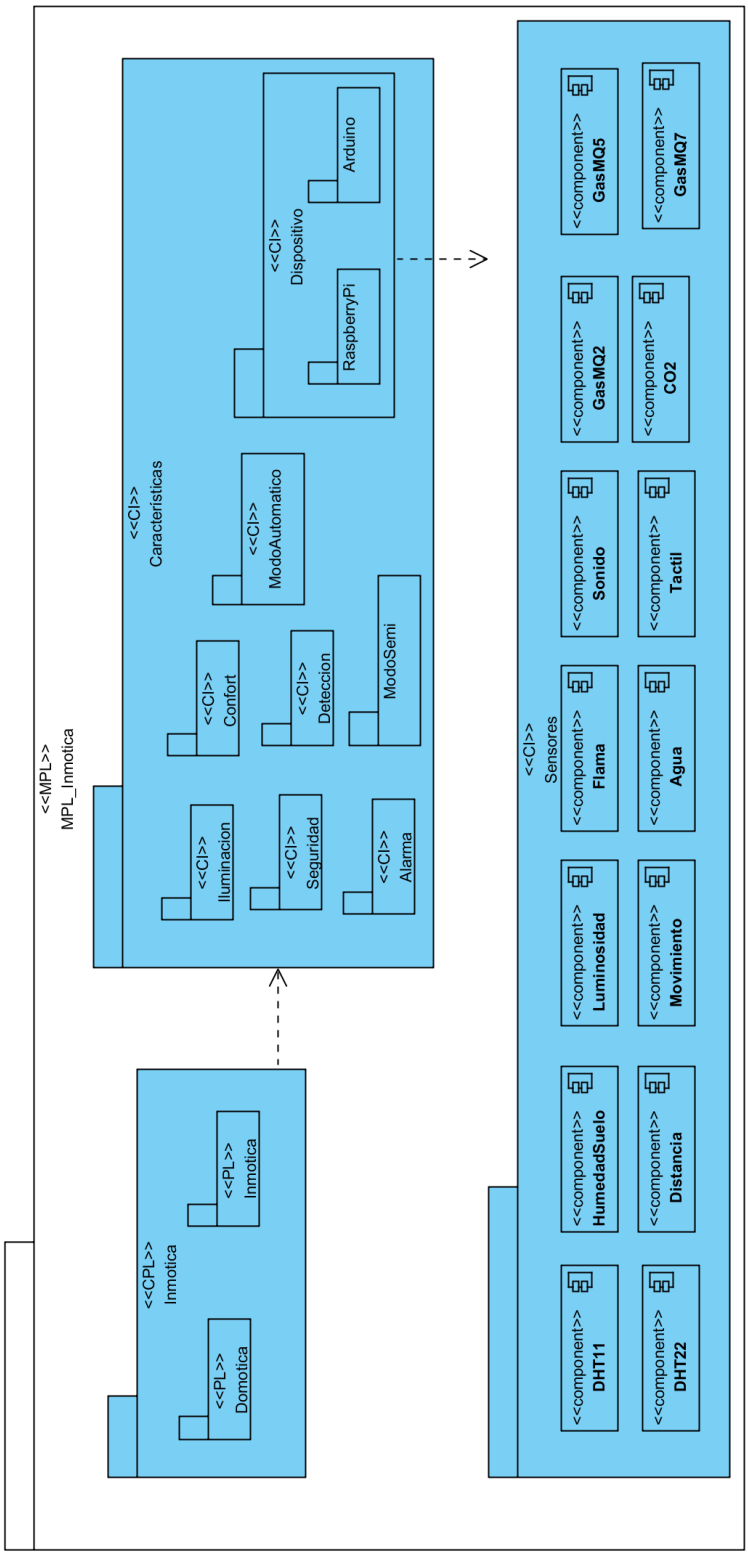


Figura 3.18: Arquitectura de Referencia de la MPL

3.9.3. Implementación de los artefactos de hardware

Para demostrar la aplicación y utilidad del modelo matemático, se propuso como caso de estudio el desarrollo de una MPL para la fabricación de software en el área de Inmótica.

Actualmente, existen diversas tecnologías para implementar sistemas domóticos e inmóticos, entre las cuales destacan Raspberry Pi (Figura 3.19) y Arduino (Figura 3.20) .

Figura 3.19: Raspberry Pi

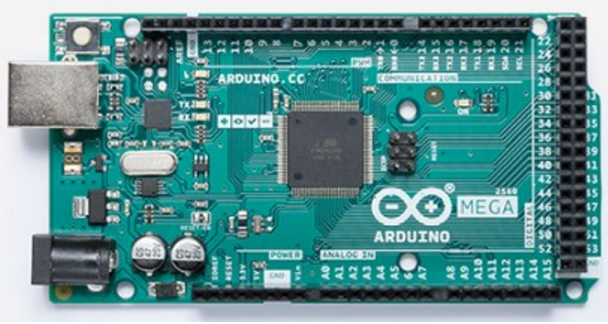


Figura 3.20: Arduino

Como parte del trabajo de tesis, se buscó maximizar la reutilización de las características de las LPS (Domótica e Inmótica) e implementar así la MPL Inmótica. Sin embargo, algunos artefactos necesarios para la implementación de la MPL, principalmente módulos para el control de dispositivos de hardware (Arduino, Raspberry Pi, sensores y actuadores) no se tenían disponibles.

Las características implementadas en los dispositivos Arduino (Figura3.21) y Raspberry (Figura 3.22) son:

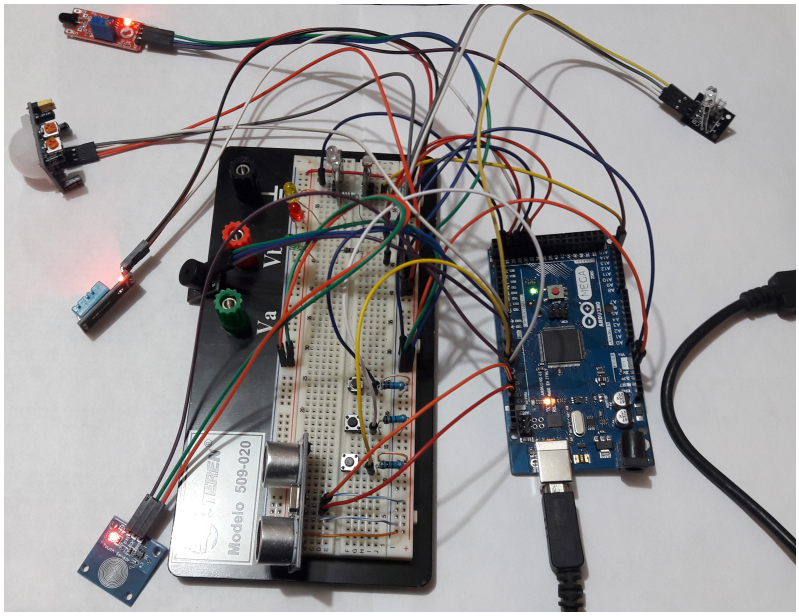


Figura 3.21: Implementación de características en Arduino

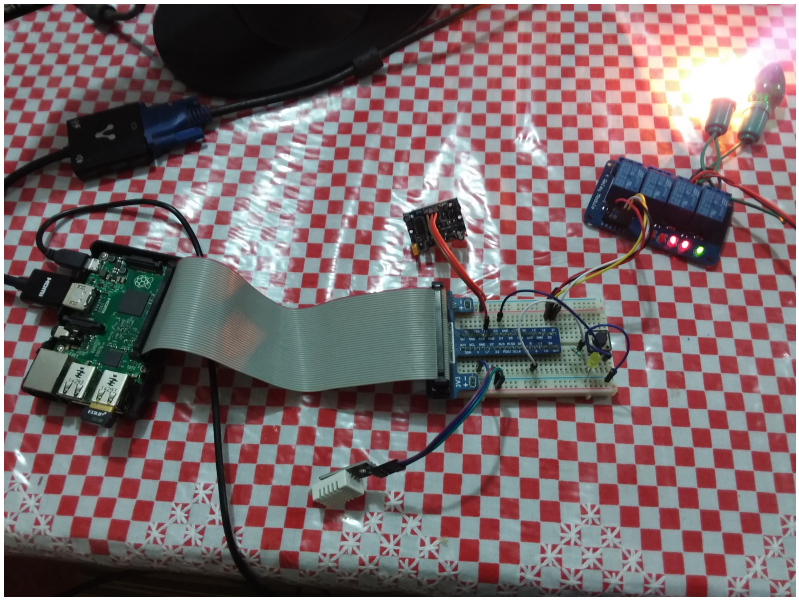


Figura 3.22: Implementación de características en Raspberry Pi

- Iluminación de forma semiautomática mediante un interruptor o una computadora.
- Iluminación de forma automática con base en el horario, detección de presencia humana o ausencia de luz (oscuridad).
- Lectura de temperatura y humedad.
- Detección de presencia.
- Detección de flama o incendios.
- Detección de humedad en el suelo (Por ejemplo: invernaderos, plantas, jardines).
- Detección de gases (CH4, CO2, Humo, entre otros).
- Activación de alarmas (silenciosa, visual o sonora) en caso de incendio o presencia de gases (Figura 8).
- Climatización de forma automática mediante un interruptor o una computadora.
- Climatización de forma semiautomática con base en el horario o presencia.
- Control de TV.

3.9.3.1. Sensor de humedad relativa y temperatura

Los sensores DHT11 y DHT22 son sensores digitales de temperatura y humedad, fáciles de implementar con cualquier microcontrolador.

- Sensor DHT11 (Figura 3.23): este sensor tiene un rango de medición de temperatura de 0 a 50 °C con precisión de ś2.0 °C y un rango de humedad de 20 % a 90 % RH con precisión de 4 % RH. Los ciclos de lectura deben ser como mínimo 1 o 2 segundos.

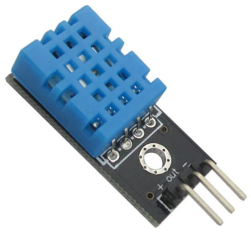


Figura 3.23: DHT11

- Sensor DHT22 (Figura 3.24): El rango de medición de temperatura es de -40 °C a 80 °C con precisión de ś0.5 °C y rango de humedad de 0 a 100 % RH con precisión de 2 % RH, el tiempo entre lecturas debe ser de 2 segundos.

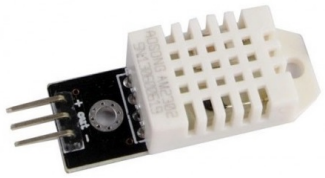


Figura 3.24: DHT22

La diferencia entre ambos sensores es solo el rango y precisión, otra diferencia es que el DHT11 soporta ciclos más rápidos de lectura.

3.9.3.2. Sensor detector de flama

Se refiere a un módulo con sensor infrarrojo que permite detectar llama (Figura 3.25). Este sensor de llama óptico conocido como detector de fuego, permite detectar la existencia de combustión por la luz emitida por fuego. Se llama fuego al conjunto de partículas o moléculas incandescentes de materia combustible, capaces de emitir luz visible, producto de una reacción química de oxidación violenta. Las llamas son las partes del fuego que emiten luz visible. El fuego conlleva un conjunto de peligros como las quemaduras o intoxicación por inhalación de humo. Este dispositivo ayuda a prevenir incendios o detectarlos de forma temprana para intentar salvar vidas y bienes económicos como estructuras, edificios, documentos, entre otros.

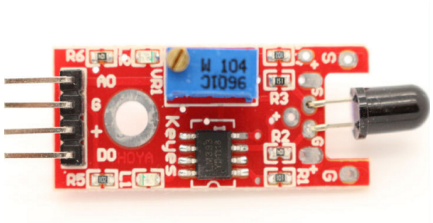


Figura 3.25: Sensor detector de flama

Características:

- Sensible a ondas entre 760-1100nm
- LED indicador de alimentación
- LED indicador de salida
- AO, salida de señal Análoga
- DO, salida de señal Digital
- Permite ajustarse por potenciómetro
- Rango de ángulo de detección: alrededor de 60 grados
- Fuente de alimentación: 0-5 V DC
- Diámetro del orificio interno: Aprox. 3mm

3.9.3.3. Sensor de nivel de agua o gotas de lluvia

El sensor de nivel de agua (Figura 3.26) produce una señal analógica proporcional a la humedad detectada, o al nivel de agua que recoja. Es un sensor de pequeñas dimensiones para detectar el agua de lluvia o bien pequeñas modificaciones de nivel de agua (un par de cm). Es útil para detectar filtraciones o fugas de líquido. Este sensor tiene bajo consumo de energía y alta sensibilidad.

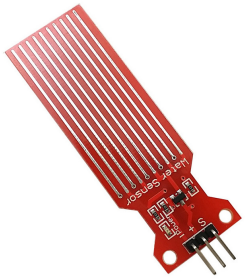


Figura 3.26: Sensor de nivel de agua

Características:

- Voltaje de funcionamiento: 5V
- Corriente de trabajo: <20 mA
- Interfaz: analógica
- Ancho de detección: 40 mm Œ 16 mm
- Temperatura de trabajo: 10 °C - 30 °C
- Señal de voltaje de salida: 0 - 4.2V

3.9.3.4. Sensor detector de ritmo cardíaco en dedo

El módulo KY-039 (Figura 3.27) es un detector de ritmo cardíaco que utiliza un LED infrarrojo brillante (IR) y un fototransistor para detectar el pulso sanguíneo en un dedo de la mano. Este monitor de pulso funciona como sigue: El LED se ubica a un lado del dedo, y el fototransistor en el otro lado del dedo. El fototransistor utiliza el flujo de Luz emitido que atraviesa el dedo para obtener el pulso cardíaco. Cuando el pulso de la presión arterial pasa por el dedo, la corriente de la base del fototransistor se modifica ligeramente debido a que la cantidad de luz que atraviesa el dedo disminuye, lo que significa una salida diferente por el puerto análogo. Es un buen detector de pulso, pero es importante considerar que es prioritario que se mantenga lo más alejado de la luz parásita, es decir de la luz externa por ejemplo la iluminación del hogar,

por eso se recomienda usar algún sistema que sirva de escudo al fototransistor y que evite que otra luz diferente a la del LED llegue, ya que la señal del latido del corazón es muy débil y la luz ambiente añadirá un ruido considerable. Es compatible con Arduino, Raspberry Pi y otros microcontroladores.

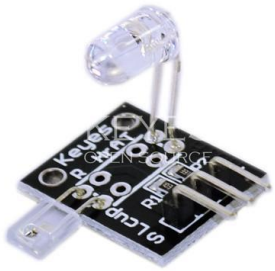


Figura 3.27: Sensor detector de ritmo cardíaco

3.9.3.5. Sensor luminoso

Las resistencias fotográficas, también conocidas como resistencias dependientes de la luz (LDR, Light-Dependent Resistor), son dispositivos sensibles a la luz que se utilizan para indicar la presencia o ausencia de luz, o para medir la intensidad de la luz ambiente. En la oscuridad, su resistencia es muy alta, a veces hasta 1 $M\Omega$, pero cuando el sensor LDR está expuesto a la luz, la resistencia disminuye drásticamente, incluso hasta unos pocos ohmios, dependiendo de la intensidad de la luz. Los LDR tienen una sensibilidad que varía con la longitud de onda de la luz aplicada y son dispositivos no lineales. El sensor LDR (Figura 3.28)es ampliamente utilizado en cámaras, lámparas de jardín y calle, detectores, relojes, luces automáticas, entre otras aplicaciones. Los valores de su resistencia, sensibilidad, coeficiente de temperatura y su curva de voltaje-corriente dependen directamente de la cantidad de luz que recibe el sensor.

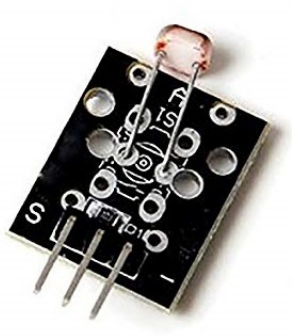


Figura 3.28: Sensor luminoso

3.9.3.6. Sensor de sonido con micrófono

El sensor KY 038 detecta el sonido proveniente de cualquier onda sonora, con esto es capaz de detectar una distancia determinada. Se asemeja al sistema de orientación por sonido que poseen los murciélagos para detectar obstáculos o presas.

Un micrófono es un transductor que convierte las ondas sonoras en señales eléctricas. KY038 es un módulo sensor de sonido basado en un micrófono que entrega una salida digital de nivel bajo cuando un sonido supera el valor prefijado con el preset. También dispone de salida analógica donde se obtiene la señal que sale directamente del micrófono. Posee 2 LEDs, uno de los cuales indica que el módulo está alimentado y el otro enciende cuando el sonido supera el nivel prefijado (útil para realizar el ajuste).

NOTA: El umbral de sensibilidad se ajusta mediante el potenciómetro en el sensor.

3.9.3.7. Sensor de gas

Los sensores de gas permiten tomar lecturas de distintos gases que estén presentes en el aire, algunos de ellos extremadamente peligrosos y/o nocivos y difícilmente detectables sin hacer uso de la tecnología, por lo que la utilidad de estos sensores es enorme.

Los sensores de gas se utilizan en lugares interiores a temperatura ambiente. La salida es una señal analógica y se lee con una entrada analógica. Es posible calibrarlos, pero se requiere de una

concentración conocida del gas a los gases a detectar. Existen distintos modelos de sensores de gas (Figura 3.29), cada uno de ellos especializado en detectar uno o varios gases determinados.

- MQ-2: Metano, Butano, Humo, Gas Licuado de Petróleo (LPG).
- MQ-3: Alcohol, Etanol, Humo.
- MQ-4: Metano, Gas Natural Comprimido o Vehicular (CNG).
- MQ-5: Gas Natural y Gas Licuado de Petróleo (LPG).
- MQ-6: Butano y Gas Licuado de Petróleo (LPG).
- MQ-7: Monóxido de Carbono.
- MQ-8: Gas de Hidrogeno.
- MQ-9: Monóxido de Carbono y Gases inflamables.
- MQ-131: Ozono.
- MQ-135: Calidad del Aire.
- MQ-136: Gas de sulfuro de hidrógeno.
- MQ-137: Amoníaco.
- MQ-138: Benceno, Propano, Alcohol, Acetona, Gas de Hidrogeno, Gas de Formaldehído, Tolueno (Metilbenceno).
- MQ-214: Metano y Gas Natural.
- MQ-216: Gas Natural y Gas de Carbón.
- MQ-303: Alcohol, Etanol, Humo.
- MQ-306: Butano y Gas Licuado de Petróleo (LPG).
- MQ-307: Monóxido de Carbono.

- MQ-309: Monóxido de Carbono y Gases inflamables.
- MG-811: Dióxido de Carbono (CO2).
- AQ-2: Humo y Gases inflamables.
- AQ-3: Alcohol y Benceno.
- AQ-7: Monóxido de Carbono.
- AQ-104: Calidad del Aire.

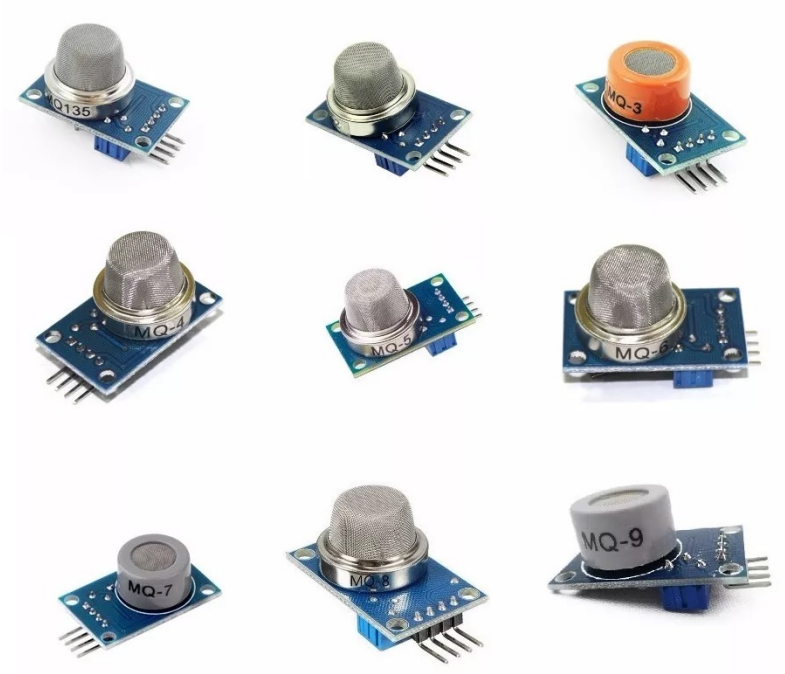


Figura 3.29: Sensores de gas

3.9.3.8. Sensor de distancia ultrasónico

El HC-SR04 (Figura 3.30) es un sensor de distancias por ultrasonido de bajo costo utilizado para detectar objetos y calcular distancias en un rango de 2 a 450cm. El sensor funciona por ultrasonidos y contiene toda la electrónica encargada de hacer la medición. El HC-SR04 se

destaca por su bajo consumo, gran precisión y bajo precio por lo que esta reemplazando a los sensores polaroid en los robots más recientes.

En robótica, estos sensores son útiles para detectar obstáculos y así tomar una decisión luego de ser detectado.

Figura 3.30: Sensor de distancia ultrasónico

3.9.3.9. Sensor higrómetro (humedad del suelo)

El sensor de humedad de suelo (higrómetro) también conocido como modulo FC-28 es un sensor que mide la humedad del suelo. Generalmente se utilizan en sistemas automáticos de riego para activar el sistema de bombeo cuando sea necesario. El sensor permite medir la conductividad del suelo, si el suelo está muy húmedo mayor será la conductividad y si el suelo está muy seco la conductividad será mucho menor. El FC-28 se distribuye con una placa de medición estándar que permite obtener la medición como valor analógico o como una salida digital, activada cuando la humedad supera un cierto umbral. Los valores obtenidos van desde 0 sumergido en agua, a 1023 en el aire (o en un suelo muy seco). Un suelo ligeramente húmero daría valores típicos de 600-700. Un suelo seco tendrá valores de 800-1023. La salida digital dispara cuando el valor de humedad supera un cierto umbral, que es posible ajustar mediante el potenciómetro. Por tanto, se obtiene una señal LOW cuando el suelo no está húmedo, y HIGH cuando la humedad supera el valor de consigna. El valor concreto dependerá del tipo de suelo y la presencia de elementos químicos, como fertilizantes. Además, no todas las plantas requieren la misma humedad, por lo que es recomendable realizar una calibración en el terreno real.

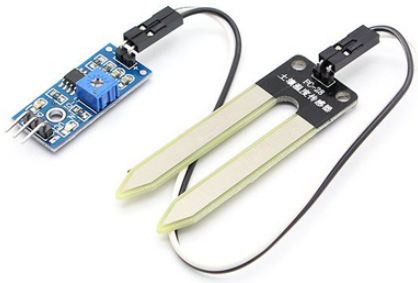


Figura 3.31: Sensor para detectar la humedad del suelo

3.9.3.10. Sensor de presencia PIR

El sensor PIR permite detectar la presencia de movimiento a través del calor. Este sensor es ideal para domótica, sistemas de alarma y detección de movimiento. El sensor HC-SR501 se basa en un sensor piroeléctrico dividido en dos zonas sensibles. Estas dos zonas sensibles en condiciones normales son activadas igualmente por la radiación infrarroja del ambiente, pero emiten una señal cuando una de ellas se encuentra en diferencia con la otra (por ejemplo, en una zona se encuentre una persona). La detección se basa en el calor emitido por el cuerpo humano, en forma de radiación infrarroja. Las zonas de detección intercaladas además se multiplican mediante un lente especial que se encuentra encima del sensor piroeléctrico (forma de cúpula).

Figura 3.32: Sensor de presencia

Características:

- Material: Semiconductor + PVC

- Voltaje: DC 5V 20V
- Corriente: 65uA
- Voltaje de salida: alto 3V / bajo 0V
- Tiempo de retardo: 0.3 18s (ajustable)
- Tiempo de bloqueo: 0.2s
- Rango de detección: 7m, en un ángulo menor a120 grados
- Temperatura de operación: -15 °C - 70 °C

3.9.3.11. Sensor táctil capacitivo

Un sensor táctil capacitivo (Figura 3.33) es un dispositivo que presenta un comportamiento similar a un pulsador. Este tipo de sensor táctil basa su funcionamiento en la medición de la variación de la capacitancia. La principal ventaja de este tipo de sensores es que no requieren de contacto físico para realizar el disparo, siendo suficiente acercar el dedo a 1-5mm del sensor. Por este motivo se les denomina dispositivos touchless. Los sensores touchless son empleados a la hora de hacer interruptores eléctricos, por ejemplo, son frecuentes en baños y garajes. También son útiles, por ejemplo, para ubicar un pulsador táctil bajo un panel interactivo, de un vinilo con artes gráficas, o integrado bajo la madera de un mueble.

Figura 3.33: Sensor táctil capacitivo

Capítulo 4

Resultados

Este capítulo tiene como finalidad mostrar los resultados obtenidos al desarrollar el modelo matemático para apoyar la derivación de productos en una Multilínea de Productos de Software utilizando técnicas de Ingeniería de Software Basada en Búsqueda.

4.1. Proceso para el desarrollo de la MPL

Para utilizar el enfoque de MPL, es necesario tener disponibles al menos dos Líneas de Productos de Software que proporcionen la descripción de la variabilidad y representen el conjunto de productos disponibles mediante un modelo de características (gráfico o textual) y sus respectivos configuradores que generen diversos artefactos de software (insumos) como requisitos, diagramas, código fuente, documentación, arquitectura, entre otros.

En la figura 4.1 se presenta la descripción gráfica del proceso de desarrollo de la Multilínea de Productos de Software propuesto para el caso de estudio en el área de Inmótica. El proceso sigue el siguiente flujo:

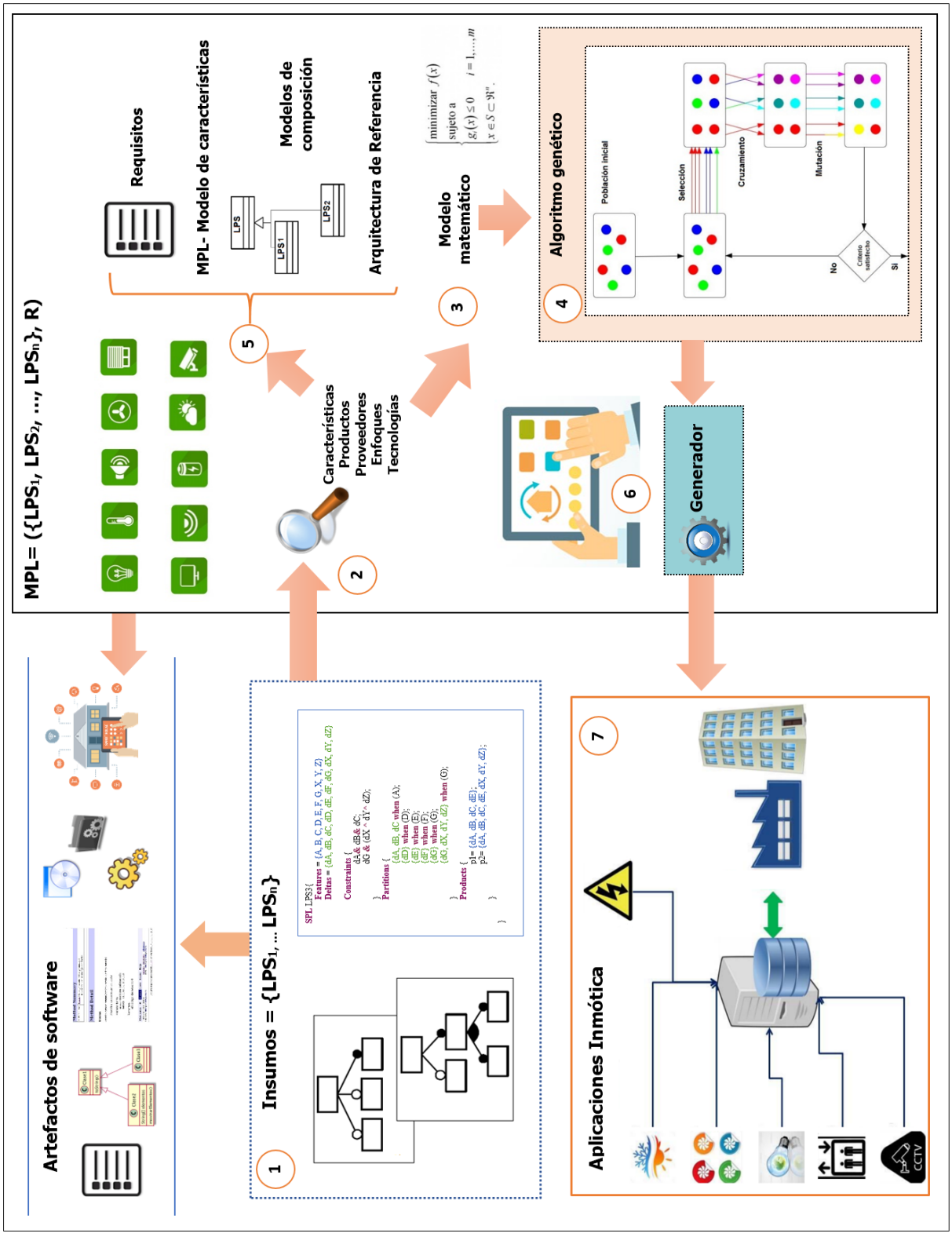


Figura 4.1: Proceso para el desarrollo de la MPL

1. Identificación de los insumos disponibles para el desarrollo de la MPL, los cuales son:

   - Línea de Productos de Software dirigida al dominio de domótica aplicando el enfoque composicional de DeltaJ y utilizando insumos de la empresa RODAS Computación S.A. de C.V.
   - Línea de Productos de Software para el dominio de inmótica aplicando MDA (Arquitectura dirigida por modelos), Scala, AspectJ y utilizando insumos de la empresa RODAS Computación S.A. de C.V.

2. Análisis de las LPS disponibles con el fin de identificar todos los aspectos y elementos que las describen principalmente en términos de proveedores, enfoques, tecnologías, características y productos (Tabla 4.1).

Tabla 4.1: Análisis de LPS disponibles

LPS	Características	Productos	Tecnologías	Enfoques
Domótica	13	6	DeltaJ, CVL	DOP
Inmótica	16	7	MDA, AspectJ, Scala	POA

3. Formulación del modelo matemático que represente la configuración de productos (selección de características) en una MPL. Para ello, primero se definió el problema y después se modeló matemáticamente (Figura 4.2).

4. Solución del modelo matemático mediante un algoritmo genético para obtener productos de software (configuraciones de características) para la automatización inteligente de edificios.

5. Desarrollo de los artefactos de software como: requisitos, modelo de características de la MPL, Arquitectura de Referencia, modelos de composición, módulos para el control de

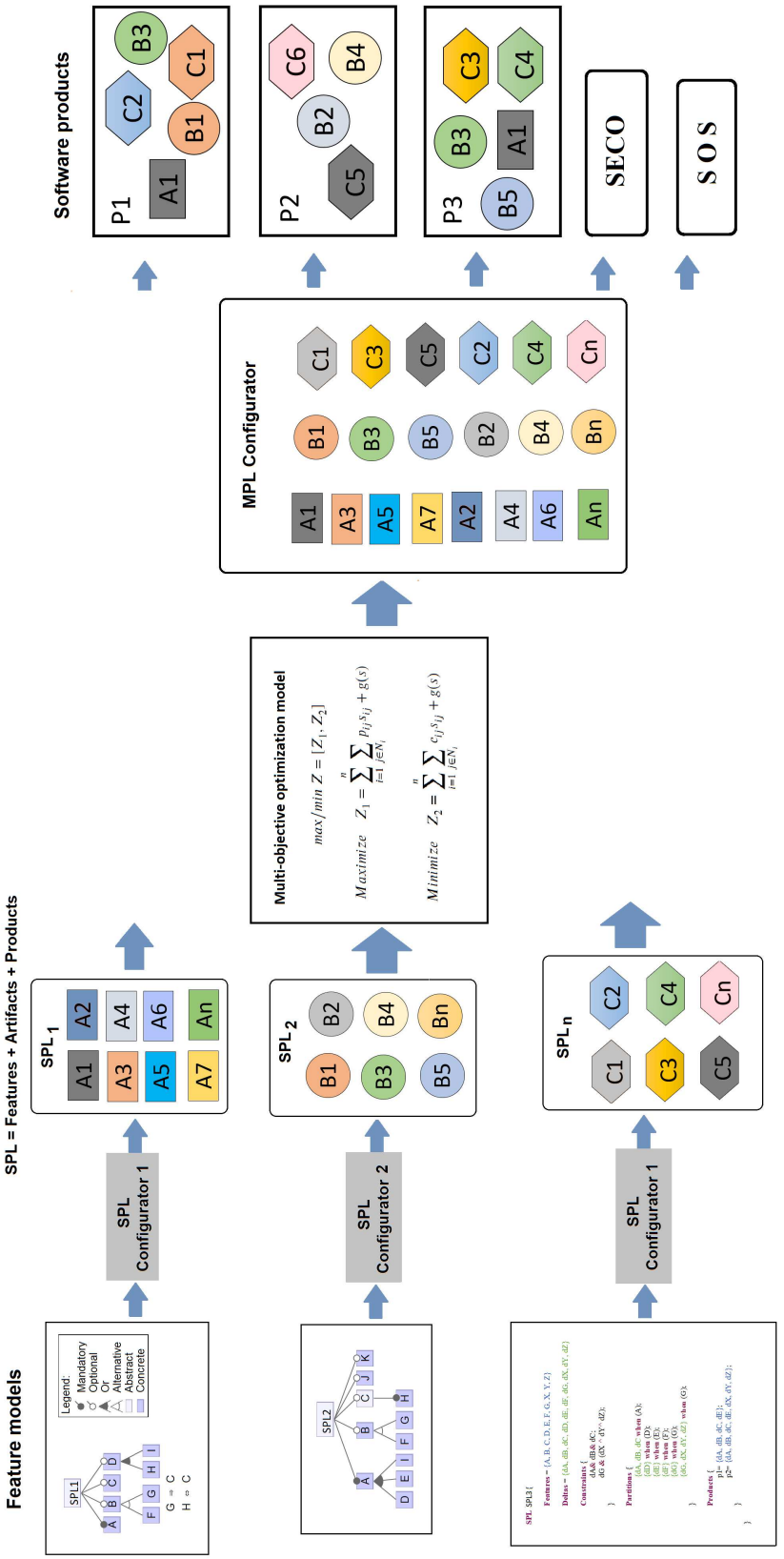


Figura 4.2: Derivación de productos en una MPL

los dispositivos de hardware (Raspberry Pi, Arduino, sensores y actuadores), entre otros activos de la MPL.

6. Desarrollo del generador (configurador), el cual es un programa que dada una especificación produce su implementación. Esta entidad se encarga del control de qué, cómo y en qué orden conectar los activos de las LPS con base en las especificaciones dadas. Asimismo, reutiliza los artefactos de la plataforma. El configurador de la MPL tiene la capacidad de generar automáticamente los siguientes artefactos: requerimientos textuales específicos de la aplicación, diagrama de clases, arquitectura de la aplicación, reporte de pruebas unitarias del cliente y servidor para la aplicación, reporte de pruebas de integración de la aplicación, ejecutables y documentación.

7. Generación de aplicaciones del área Inmótica de forma automática utilizando el generador.

4.2. Método para gestionar la variabilidad

El método propuesto para gestionar la variabilidad de la MPL y generar una nueva cartera de productos se presenta en la Figura 4.3. Este método se basa en el análisis y la comparación de modelos de características para encontrar coincidencias entre las características maximizando la compatibilidad y la reutilización de las características al menor costo posible. El método propuesto tiene seis pasos principales:

1. Identificación de la variabilidad. Como primer paso, es necesario identificar la variabilidad disponible en las LPS, que consiste en enumerar las características comunes y variables de los productos y sus relaciones (obligatorio, opcional y alternativo) para cada LPS.

2. Evaluación de las LPS que integrarán el MPL. Este paso se refiere a evaluar la capacidad de reutilización y compatibilidad de las características de cada LPS que integrará el MPL. En este paso, los modelos de características se traducen a los datos requeridos por el modelo de optimización y se realizan los cálculos necesarios como r_{ij}, p_{ij} y c_{ij}.

3. Implementación del modelo de optimización multi-objetivo propuesto en MULTIGEN.

4. Resolución del modelo de optimización multi-objetivo con los algoritmos genético incluidos en MULTIGEN.

5. Comparación de los resultados obtenidos por cada algoritmo genético y elegir la configuración óptima.

6. Creación del modelo de características de la MPL. Con los resultados obtenidos del modelo de optimización multi-objetivo, crear un nuevo modelo de características (MPL-Feature Model). Fusionar las características con alto nivel de reutilización a través de técnicas de composición.

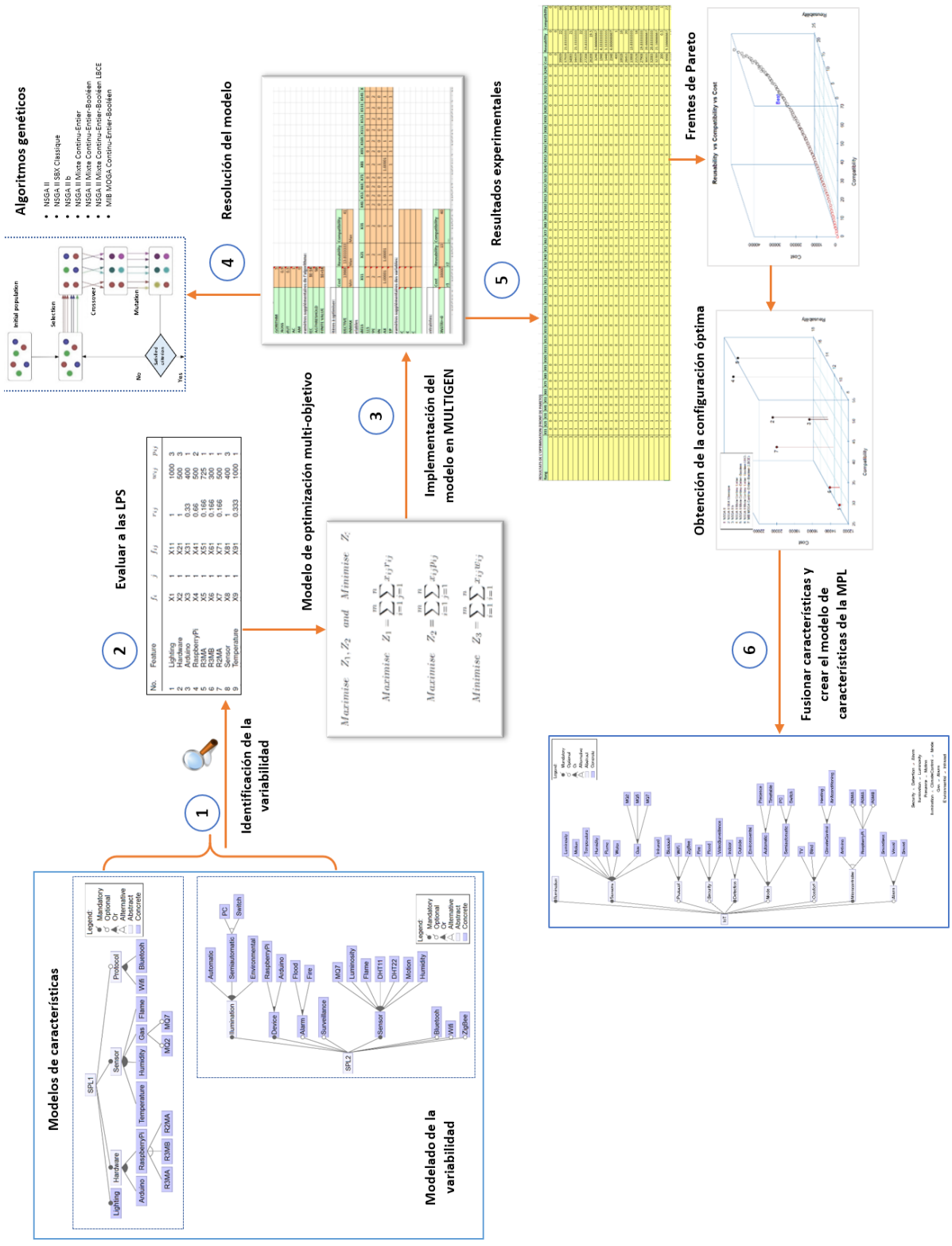


Figura 4.3: Método propuesto

4.3. Arquitectura de Referencia

En esta sección, se describe la aplicación del enfoque Archample a la MPL de Inmótica.

4.3.1. Preparación

En esta fase, se identificó a las partes interesadas y al equipo de evaluación como se define en la Sección 3.6.

4.3.2. Selección de la posible descomposición de MPL

Para el proyecto de la MPL Inmótica se identificaron cuatro alternativas diferentes de arquitectura MPL:

1. Una LPS: Define el sistema como una LPS.
2. Dos LPS: Define el sistema como dos LPS independientes (Sección 3.9.2 Figuras 3.16 y 3.17).
3. AD LPS: Define solo los dominios de aplicación como LPS.
4. CPL: Define una composición de LPS (Sección 3.9.2 Figura 3.18.

Se llevó a cabo la evaluación de cuatro alternativas diferentes utilizando el enfoque de evaluación GQM, como parte de Archample.

La tabla 4.2 muestra los resultados de GQM definidos durante la evaluación. Los objetivos representan objetivos comerciales de alto nivel que se consideraron importantes para los responsables de la toma de decisiones del proyecto.

Tabla 4.2: Resultados de GQM para la arquitectura MPL

Objetivo	Pregunta	Métrica
Maximizar la reutilización	¿Cuál es el nivel de reutilización de los activos?	Reutilización de la característica
Minimizar el costo de desarrollo	¿Cuál es el costo de desarrollo de los activos?	Costo de la característica
Maximizar la compatibilidad de las características	¿Cuál es el puntaje positivo o negativo que se le da a la característica por la capacidad de integrarse con otra característica?	Evaluación subjetiva por gestión de proyectos

Una vez que se definieron los objetivos, las preguntas y las métricas, se comenzó con la evaluación real.

4.3.3. Evaluación de la alternativa de diseño de la MPL

Después de seleccionar la composición de LPS como la alternativa de diseño más factible para la MPL, fue necesaria evaluar las características de las LPS. En este caso, se propone utilizar el modelo de optimización multi-objetivo (MPL-FES) para evaluar la selección de características como una extensión al enfoque ARCHAMPLE (Sección 4.4.2.2).

4.3.4. Informes y taller

El diseño completo y la evaluación correspondiente de las alternativas de MPL se presenta en la Sección 4.4.4.2.

4.4. Modelo matemático

La selección óptima de características en una MPL se identifica como un problema de tiempo polinomial no determinista (NP-duro). Para apoyar la derivación de productos en una MPL, es

necesario definir el problema de la configuración de productos. En esta sección, primero se abordó el problema de la configuración de productos en las LPS y luego se modeló el problema en una MPL como un problema de búsqueda.

4.4.1. Línea de Productos de Software

Las variables que intervienen en el proceso de configuración de una LPS son:

- La decisión de seleccionar o no una característica, se modela por la variable binaria $x_i \in \{0,1\}$.
- Las constantes definidas por el modelo son las siguientes:
  - $T = (V,A)$ es un árbol de características donde V es un conjunto de nodos (características) y A es un conjunto de arcos tal que $(i, j) \in A$ denota que la característica j es seleccionada si la característica i también es seleccionada.
  - $M \subseteq V$ es el conjunto de características obligatorias, tal que el nodo raíz $r \in M$.
  - $O \subset V$ es el conjunto de características opcionales, tal que $O \cup M = V$ y $O \cap M = 0$.
  - $XOR_i \subset 2^V$ es el conjunto de todas las características alternativas en $i \in V$ tal que si $\exists P \in XOR_i$ entonces $(i,s) \in A$, para todo $s \in P$. Además, $XOR_i = 0$ si no existe una alternativa exclusiva basada en i.
  - $OR_i \subset 2^V$ es el conjunto de todas las características alternativas no exclusivas en $i \in V$ tal que si $Q \in OR_i$ entonces $(i,s) \in A$, para todo $s \in Q$. Además, $OR_i = 0$ si no existe una alternativa no exclusiva basada en i.
- Las constantes que describen los requerimientos no funcionales:
  - $c_i \in \mathbb{R}$ es el costo de la característica $i \in V$.
  - $D_i \in \mathbb{R}$ es el presupuesto disponible del cliente.
  - $b_i \in \mathbb{R}$ es el grado de preferencia de la característica $i \in V$, con $b_i = 0$ para todo $i \in M$, y $b_j \leq b_i$, para todo $(i, j) \in A$.

- $E = (E_1(x), E_2(x), \ldots)$ es el conjunto de las restricciones, donde $E_j(x)$ es una expresión lógica sobre las variables x.

Dadas las variables y constantes descritas arriba, la función objetivo que describe la configuración de productos en una LPS se define a continuación (ecuación 4.1):

$$Maximizar \quad F(x) = \sum_{i \in V} b_i x_i \tag{4.1}$$

Sujeto a:

$$\sum_{i \in V} c_i x_i \leq D \tag{4.2}$$

El conjunto de restricciones del modelo son representados por las ecuaciones (4.3, 4.4, 4.5, 4.6, 4.7).

La restricción 4.3 asegura que la característica obligatoria $i \in M$ es seleccionada si el padre $p \in V$ es seleccionada.

$$x_p = 1 \rightarrow x_i = 1 \qquad (p,i) \in A : i \in M \tag{4.3}$$

La restricción 4.4 asegura que la característica opcional i no es seleccionada si el padre $p \in V$ no se selecciona.

$$x_p = 0 \rightarrow x_i = 0 \qquad (p,i) \in A : i \in O \tag{4.4}$$

La restricción 4.5 asegura que por cada conjunto alternativo exclusivo $P \in XOR_p$, exactamente una característica en P es seleccionada si p es seleccionada.

$$x_p = 1 \rightarrow \sum_{i \in P} x_i = 1 \qquad \forall p \in V : XOR_p \neq 0, \forall P \in XOR_p \tag{4.5}$$

La restricción 4.6 asegura que por cada conjunto alternativo no exclusivo $Q \in OR_p$, al menos una característica en Q es seleccionada si p es seleccionada.

$$x_p = 1 \rightarrow \sum_{i \in Q} x_i \geq 1 \qquad \forall p \in V : OR_p \neq 0, \forall Q \in OR_p \tag{4.6}$$

La restricción 4.7 asegura que las restricciones del modelo de características se satisfacen.

$$E_j(x) = true \qquad \forall E_j(x) \in E \tag{4.7}$$

4.4.2. Multilínea de Productos de Software

La configuración de una MPL es un problema de optimización combinatoria no lineal y multi-objetivo [1] en el que se involucran un conjunto finito de Líneas de Productos de Software, una cierta cantidad de características y un conjunto finito de restricciones. Cada característica tiene diferentes valores y pertenece a diferentes modelos de características o Líneas de Productos de Software y cuyo número de combinaciones crece exponencialmente (ver 1.1.2.6), dando lugar a múltiples configuraciones (soluciones óptimas) para un producto de software. [2] La problemática de configuración de productos de software en una MPL se abordó bajo un enfoque de optimización multi-objetivo considerando el antagonismo de los criterios técnicos y económico para la selección de características (costo, compatibilidad y reutilización) facilitando así la toma de decisiones al momento de elegir las posibles configuraciones.

4.4.2.1. Formalización del problema

Dado *m* modelos de características y una especificación de requisitos, una configuración de productos es la selección de características subdivididas en *m* LPS. Las características seleccionadas deben ser compatibles entre sí y la configuración debe cumplir con los requisitos de la aplicación y optimizar la reutilización de los artefactos de la LPS disponibles. En consecuencia,

[1] Un problema multi-objetivo difiere de un problema mono-objetivo, dado que requiere la optimización simultánea de más de una función objetivo en paralelo. En este contexto, la noción de óptimo se redefine porque en lugar de buscar una única mejor solución (solución exacta), se busca un conjunto de buenas soluciones. El desafío principal de los algoritmos de optimización multi-objetivo es encontrar un conjunto de soluciones para ofrecer al tomador de decisiones las mejores alternativas entre las disponibles, para que seleccione una de ellas.

[2] Este fenómeno se denomina explosión combinatoria, debido a la gran cantidad de soluciones factibles e infactibles que aparecen. En la práctica, las técnicas exactas no son aplicables para su solución debido al gran tiempo de cómputo necesario. Estos problemas se conocen como problemas tipo NP (nondeterministic polynomial time) completo y no es posible encontrar un algoritmo que los resuelva en un tiempo de cómputo polinomial.

si un producto de software en particular se define por n características, con cada característica $f_{ij} \in N_i$, la cual tiene un costo por reutilización de r_{ij}, una recompensa de selección pij por compatibilidad y un costo de desarrollo w_{ij}, es posible modelar el problema de configuración de productos en una MPL como un caso especial de MCKP (Multiple Choice Knapsack Problem).

Para formular el problema matemáticamente, la notación empleada se muestra en la Tabla 4.3.

Tabla 4.3: Notación

Notación	Descripción
f	Característica
n	Número de características
m	Número de LPS. Variable igual al número de modelos de características.
$i = 1, 2, ..., n$	Índice de una característica
$j = 1, 2, ..., m$	Índice de una LPS
x_{ij}	Variable igual a 1 si la característica i se asigna a la LPS j; 0 en caso contrario.
r_{ij}	Reutilización de la característica i si se asigna a la LPS j
w_{ij}	Costo de desarrollo de la característica i si se asigna a la LPS j
p_{ij}	Beneficio o puntaje de la característica i si se asigna a la LPS j por compatibilidad
W	Presupuesto disponible del usuario
R	Valor límite de reutilización requerido por el usuario
MA	Conjunto de características obligatorias
O	Conjunto de características opcionales
XOR	Conjunto de todas las características alternativas exclusivas
OR	Conjunto de todas las características alternativas no exclusivas

- Todos los coeficientes r_{ij}, p_{ij}, w_{ij}, W y R son números positivos.
- La reutilización R_{ij} (Ecuación 4.8) define la frecuencia de usoU_{ij} (Equación 4.9) de una característica i en un modelo de características j.

$$R_{ij} = \sum_{i=1}^{n} \frac{U_{ij}}{n} \tag{4.8}$$

$$U_{ij} = \sum_{i=1}^{n} U_{ij} \tag{4.9}$$

4.4.2.2. Modelo de optimización multi-objetivo

En conclusión, el problema de la selección de características o configuración de productos en una Multilínea de Productos de Software con restricciones obligatorias y alternativas se formaliza en la Ecuación 4.10. El modelo de optimización multi-objetivo propuesto integra y optimiza simultáneamente tres funciones objetivo: reutilización del software (Z_1), compatibilidad del software (Z_2) y costo (Z_3) con el mismo conjunto de características.

La función objetivo (Ecuación 4.11) es una optimización por maximización, la cual tiene como objetivo seleccionar características de cada LPS con el máximo beneficio total (reutilización), mientras que el costo de desarrollo de las características elegidas no debe exceder la restricción de presupuesto W (Ecuación 4.14).

La función objetivo (Ecuación 4.12) es una optimización por maximización, la cual tiene como objetivo seleccionar características de cada LPS con la máxima compatibilidad, mientras que el costo de desarrollo de las características elegidas no debe exceder la restricción de presupuesto W (Ecuación 4.14).

La función objetivo (Ecuación4.13) es una optimización por minimización, la cual tiene como objetivo seleccionar características de cada LPS con el costo mínimo de desarrollo, mientras que la reutilización debe exceder el valor límite de reutilización (Ecuación 4.15).

$$Maximizar \quad Z_1, Z_2 \quad and \quad Minimizar \quad Z_3 \tag{4.10}$$

$$Maximizar \quad Z_1 = \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij} r_{ij} \tag{4.11}$$

$$Maximizar \quad Z_2 = \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij} p_{ij} \tag{4.12}$$

$$Minimizar \quad Z_3 = \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij} w_{ij} \tag{4.13}$$

Sujeto a

$$\sum_{i=1}^{m} \sum_{j \in N_i} w_{ij} x_{ij} \leq W, \tag{4.14}$$

$$\sum_{i=1}^{m} \sum_{j \in N_i} r_{ij} x_{ij} \geq R, \tag{4.15}$$

$$\sum_{j \in N_i} x_{ij} = 1, \quad \forall i \in \{1,2 \ldots n\} \tag{4.16}$$

$$x_{ij} \in \{0,1\}, \quad \forall i = \{1,2 \ldots n\} \quad \forall j = \{1,2 \ldots m\}, x_{ij} \in N_i \tag{4.17}$$

$$xp_{=}1 \rightarrow x_{ij} = 1 \quad i \in MA \tag{4.18}$$

$$x_p = 0 \rightarrow x_{ij} = 0 \quad i \in O \tag{4.19}$$

$$x_p = 1 \rightarrow \sum_{i \in P} x_{ij} = 1 \qquad \forall P \in XOR_p \tag{4.20}$$

$$x_p = 1 \rightarrow \sum_{i \in Q} x_{ij} \geq 1 \qquad \forall Q \in OR_p \tag{4.21}$$

La restricción (Ecuación 4.16) garantiza la selección de una sola característica de cada LPS j.

La restricción (Ecuación 4.17) garantiza que cada característica $f_{ij} \in N$ tenga asociada una variable de decisión $x_{ij} \in \{0,1\}$, de manera que $x_{ij} = 1$ si la característica i se asigna a la LPS

j y se incluirá en el producto de software final. De lo contrario, $x_{ij} = 0$ si esta característica no está incluida en el producto.

La restricción (Ecuación 4.18) garantiza que la característica obligatoria $ij \in MA$ es seleccionada si la característica padre es seleccionada.

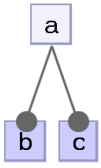
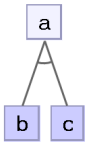
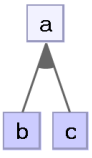
La restricción (Ecuación 4.19) garantiza que la característica opcional $ij \in O$ no se selecciona si la característica padre no está seleccionada.

La restricción (Ecuación 4.20) garantiza que para cada conjunto de características XOR, se seleccione exactamente una característica en P si se selecciona p.

La restricción (Ecuación 4.21) garantiza que para cada conjunto de funciones OR, se seleccione al menos una característica en Q si se selecciona p.

El conjunto de restricciones descritas por el modelo de características se describe en la Tabla 4.4.

Tabla 4.4: Restricciones del modelo de características

Modelo de características (Ejemplo)	Significado
Obligatoria 	a requiere b y c
Alternativa exlusiva 	a requiere b o c, pero no ambos
Alternativa no exclusiva 	a requiere b o c, o ambos

Continúa en la página siguiente

Tabla4.4 Restricciones del modelo de características (continuación)

Modelo de características (Ejemplo)	Significado
Opcional 	b es una característica opcional de a

4.4.2.3. Solución del modelo matemático

La selección óptima de características en una MPL se identifica como un problema de tiempo polinomial no determinista (NP-duro).

En esta investigación, los algoritmos genéticos se emplearon para resolver el modelo de optimización multi-objetivo debido a su buen rendimiento para manejar la explosión combinatoria de manera efectiva. Asimismo, los algoritmos genéticos también se han explorado en varios aspectos de las LPS para resolver problemas como la selección de características, pruebas, manejo de variabilidad, ubicación de características, entre otros. El algoritmo 1 muestra el proceso general del algoritmo genético.

Generar la población inicial;

Evaluar la función fitness;

while *condición de término is not true* **do**

> Selección;
>
> Cruzamiento;
>
> Mutación;
>
> Evaluar la función fitness;

end

Algorithm 1: Algoritmo genético

El modelo de optimización multi-objetivo se evaluó utilizando varios algoritmos genéticos incluidos en la biblioteca MULTIGEN [143], un complemento de Microsoft Excel. Esta bibliote-

ca se desarrolló por el Departamento de Optimización de Procesos del Laboratorio de Ingeniería Química del Institut National Polytechnique Toulouse (INP). MULTIGEN incluye 8 algoritmos (Tabla 4.5) que se distinguen por su estructura y tipo de variables (continua, entera o binaria).

Tabla 4.5: Algoritmos

Algoritmo	Tipo de variable	Descripción
NSGA-II	Continua	Algoritmo genético multi-objetivo elitista
NSGA-IIb	Continua	Algoritmo genético multi-objetivo elitista
NSGA-II MI	Continua, Entera	Algoritmo genético multi-objetivo elitista mixto
NSGA-II MIB	Continua, Entera, Binaria	Algoritmo genético multi-objetivo elitista mixto
MOGA	Binaria	Algoritmo genético multi-objetivo con representación binaria

La representación de un cromosoma utilizado en el algoritmo genético se ilustra mediante un ejemplo como se muestra en la Figura 4.4.

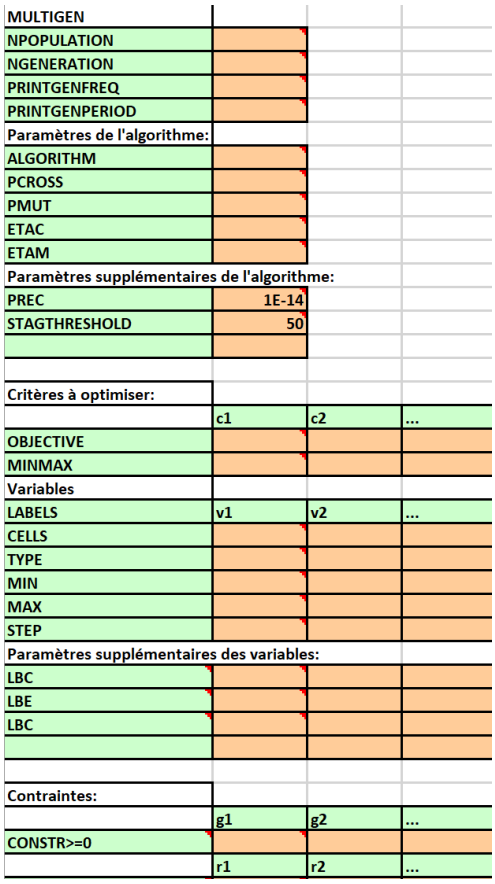
Figura 4.4: Representación de un cromosoma

4.4.3. Implementación del modelo

En esta sección se describe la búsqueda de la configuración de productos óptima mediante la resolución del modelo de optimización multi-objetivo.

4.4.3.1. Paso 1. Generación de la interfaz

El primer paso consiste en generar la interfaz de MULTIGEN que se utiliza para codificar el problema y el modelo de optimización multi-objetivo (Figura 4.5).



MULTIGEN			
NPOPULATION			
NGENERATION			
PRINTGENFREQ			
PRINTGENPERIOD			
Paramètres de l'algorithme:			
ALGORITHM			
PCROSS			
PMUT			
ETAC			
ETAM			
Paramètres supplémentaires de l'algorithme:			
PREC	1E-14		
STAGTHRESHOLD	50		
Critères à optimiser:			
	c1	c2	...
OBJECTIVE			
MINMAX			
Variables			
LABELS	v1	v2	...
CELLS			
TYPE			
MIN			
MAX			
STEP			
Paramètres supplémentaires des variables:			
LBC			
LBE			
LBC			
Contraintes:			
	g1	g2	...
CONSTR>=0			
	r1	r2	...

Figura 4.5: Interfaz de MULTIGEN

4.4.3.2. Paso 2. Introducción de los datos y parámetros

Posteriormente, en la interfaz generada se introducen los datos del caso de estudio y los parámetros del algoritmo genético elegido (Figura 4.6). Para mostrar los resultados obtenidos con la optimización del modelo, se configuró la impresión de resultados cada 50 generaciones.

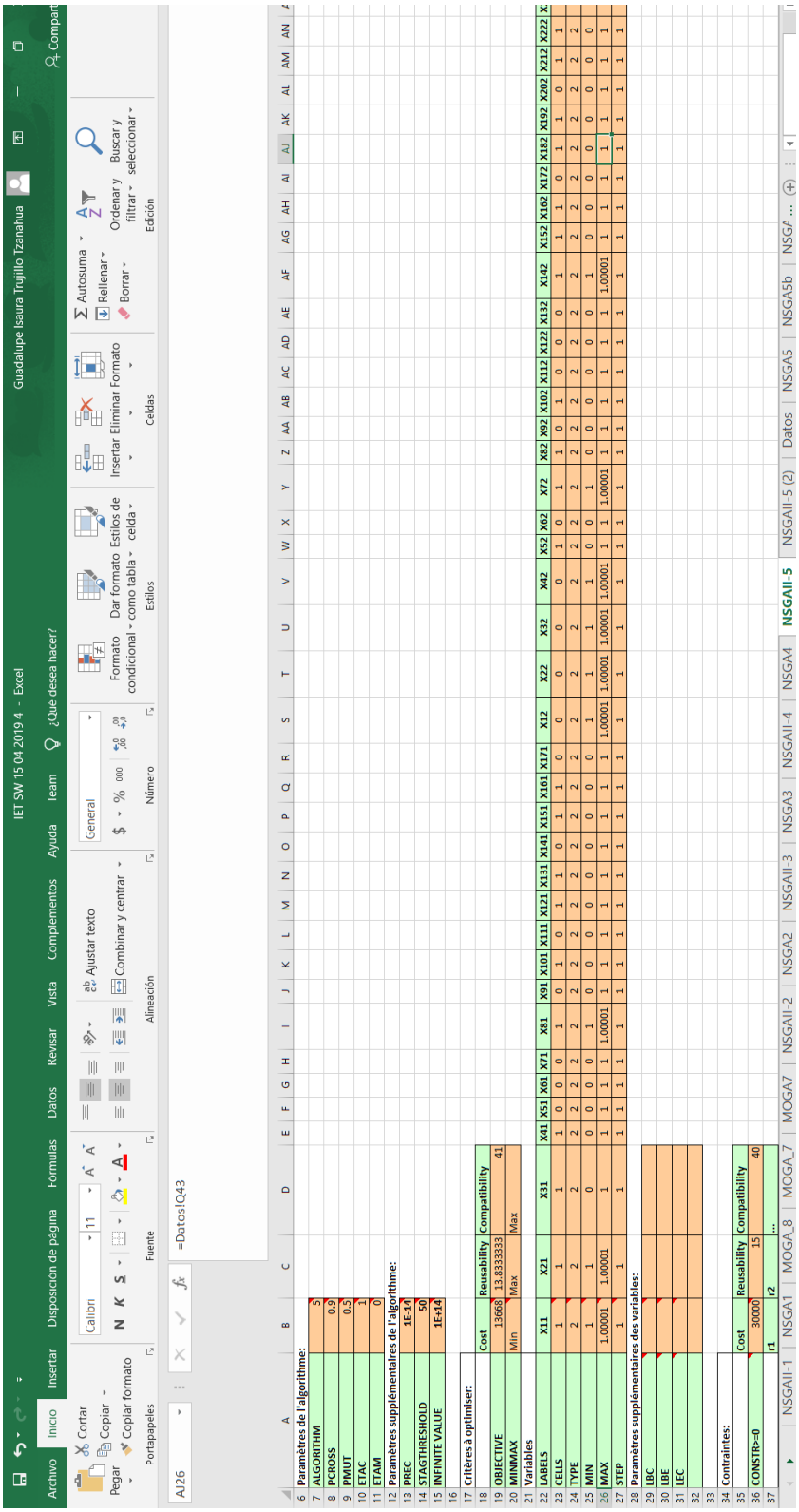


Figura 4.6: Introducción de datos en MULTIGEN

La Tabla 4.6 proporciona una muestra del conjunto de datos de características para generar una cartera de productos para la MPL.

Tabla 4.6: Datos

No.	**Característica**	f_i	j	f_{ij}	r_{ij}	w_{ij}	p_{ij}	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**
1	Lighting	X1	1	X11	1	1000	3	✓	✓	✓	✓	✓	✓
2	Hardware	X2	1	X21	1	500	3	✓	✓	✓	✓	✓	✓
3	Arduino	X3	1	X31	0.33	400	1	✓					✓
4	RaspberryPi	X4	1	X41	0.66	500	2		✓	✓	✓	✓	
5	R3MA	X5	1	X51	0.166	725	1		✓				
6	R3MB	X6	1	X61	0.166	300	1			✓			
7	R2MA	X7	1	X71	0.166	500	1				✓		
8	Sensor	X8	1	X81	1	400	3	✓	✓	✓	✓	✓	✓
9	Temperature	X9	1	X91	0.333	1000	1	✓					✓
10	Humidity	X10	1	X101	0.666	800	2	✓	✓	✓			✓
11	Gas	X11	1	X111	0.166	500	1	✓					
12	MQ2	X12	1	X121	0.5	200	1	✓	✓				✓
13	MQ7	X13	1	X131	0.166	300	1	✓					
14	Flame	X14	1	X141	0.166	1650	1	✓					
15	Protocol	X15	1	X151	0.166	320	1	✓					
16	Wifi	X16	1	X161	0.666	298	2	✓	✓	✓		✓	
17	Bluetooh	X17	1	X171	0.5	2356	1		✓		✓		✓
18	Ilumination	X1	2	X12	1	2250	3	✓	✓	✓	✓	✓	✓
19	Automatic	X2	2	X22	0.5	3500	1	✓	✓		✓		
20	Semiautomatic	X3	2	X32	0.666	1500	2	✓		✓		✓	✓
21	Environmental	X4	2	X42	0.333	2000	1	✓		✓			
22	Computer	X5	2	X52	0.666	1000	2	✓	✓	✓		✓	
23	Switch	X6	2	X62	1	2000	3	✓	✓	✓	✓	✓	✓
24	Device	X7	2	X72	1	1000	3	✓	✓	✓	✓	✓	✓
25	RaspberryPi	X8	2	X82	0.666	900	2			✓	✓	✓	✓
26	Arduino	X9	2	X92	0.333	800	1	✓	✓				
27	Alarm	X10	2	X102	0.666	1000	2	✓		✓		✓	✓

Continúa en la página siguiente

Tabla4.6 Datos (continuación)

No.	**Característica**	f_i	j	f_{ij}	r_{ij}	w_{ij}	p_{ij}	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**
28	Flood	X11	2	X112	0.5	860	1	✓		✓		✓	
29	Fire	X12	2	X122	0.5	950	1	✓		✓			✓
30	Surveillance	X13	2	X132	0.666	2500	2	✓	✓	✓	✓		
31	Sensor	X14	2	X142	1	1000	3	✓	✓	✓	✓	✓	✓
32	MQ7	X15	2	X152	0.333	500	1	✓		✓			
33	Luminosity	X16	2	X162	1	1200	3	✓	✓	✓	✓	✓	✓
34	Flame	X17	2	X172	0.333	980	1	✓	✓				
35	DHT11	X18	2	X182	0.5	250	1	✓		✓		✓	
36	DHT22	X19	2	X192	0.666	300	2		✓	✓	✓		✓
37	Motion	X20	2	X202	0.333	700	1	✓		✓			
38	Humidity	X21	2	X212	0.5	800	1	✓		✓		✓	
39	Bluetooh	X22	2	X222	0.666	1000	2	✓	✓	✓			✓
40	Wifi	X23	2	X232	0.5	1200	1	✓			✓		✓
41	ZigBee	X24	2	X242	0.166	2000	1					✓	

La Tabla 4.7 muestra los parámetros de entrada utilizados para la ejecución de cada algoritmo genético. Estos parámetros influyen en la diversidad de la población. Por ejemplo, una tasa de cruce excesivamente alta hará que la solución converja rápidamente antes de que se encuentre el óptimo. Por otro lado, una tasa de cruce baja disminuye la diversidad de la población y da como resultado un largo tiempo de cálculo.

La tasa de mutación también influye en el rendimiento del AG, ya que determina la frecuencia de búsqueda aleatoria. En general, se recomienda una tasa de mutación muy baja para evitar que el proceso del algoritmo genético se convierta en una búsqueda aleatoria pura, lo que perjudica la propiedad del AG. El tamaño de la población es el factor más distintivo que influye en la diversidad de la población.

Tabla 4.7: Parámetros del algoritmo genético

Parámetro	Descripción	Valor
Población	Número de individuos en la población	200
Generación	Número de generaciones	500
Tasa de cruzamiento	Rango (0.1 - 1.0)	0.9
Tasa de mutación	Rango (0.1 - 1.0)	0.5

4.4.3.3. Paso 3. Definición de los criterios de optimización

Después se definen los criterios a optimizar (Tabla 4.8), es decir, las tres funciones objetivo (reutilización, compatibilidad y costo), así como el tipo de función objetivo (maximizar o minimizar), las características de las variables (celda, tipo: continua, entera o binaria, valor mínimo y valor máximo) y restricciones del modelo de optimización [3].

Tabla 4.8: Funciones objetivo

Criterio a optimizar	Costo	Reutilización	Compatibilidad
Objetivo	13668	13.8333333	41
MinMax	Min	Max	Max

4.4.3.4. Paso 4. Resolución el modelo

Finalmente se resuelve el problema de optimización que consiste en encontrar el valor que deben tomar las variables para hacer óptima la función objetivo satisfaciendo el conjunto de restricciones. En este caso, se busca la mejor selección de características (configuración del producto) para desarrollar una MPL Inmótica.

[3]Para facilitar la lectura se presenta los títulos de la tabla en español, ya que el idioma original de la biblioteca MULTIGEN es el francés

4.4.4. Resultados computacionales

A continuación se presenta los resultados empíricos de los experimentos que se realizaron para evaluar los algoritmos.

4.4.4.1. Resultados experimentales

Los algoritmos se implementaron en MULTIGEN y se probaron en una PC con un procesador Intel(R) Core (TM) i7 a 2,8 GHz, 16 GB de RAM y sistema operativo Microsoft Windows 10. Para evaluar el rendimiento del modelo de optimización multi-objetivo, es necesario seleccionar el mejor enfoque para identificar soluciones candidatas. Evaluamos todos los algoritmos genéticos disponibles en la biblioteca MULTIGEN utilizando las siguientes métricas: valor de conveniencia para la reutilización, compatibilidad, costo y complejidad del tiempo.

El modelo de optimización multi-objetivo propuesto se ejecutó un total de ocho veces bajo los mismos parámetros. Para cada ejecución, los valores iniciales de las variables de optimización se cambiaron para iniciar la búsqueda desde diferentes puntos y observar si la evolución se comporta de la misma manera. Las ocho ejecuciones del modelo dieron resultados mayores que los requeridos, por lo cual es posible concluir que el modelo se desempeña adecuadamente en el escenario de optimización propuesto.

Se propuso encontrar la configuración de las características que permitirían generar un producto de software con un costo límite de $30,000.00 (presupuesto), una reutilización mínima de 15 y una compatibilidad mínima de 40.

La Figura 4.7 muestra un ejemplo de los Frentes de Pareto obtenidos con las pruebas realizadas en MULTIGEN por cada algoritmo. Los resultados indican con un 1 si la característica se selecciona y con un 0 si no se selecciona.

Resultados de la optimización (Frentes de Pareto)

Rang	X11	X21	X31	X41	X51	X61	X71	X81	X91	X101	X111	X121	X131	X141	X151	X161	X171	X12	X22	X32	X42	X52	X62	X72	X82	X92	X102	X112	X122	X132	X142	X152	X162	X172	X182	X192	X202	X212	X222	X232	X242	Cost	Reusability	Compatibility
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	39939	22	66
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	37939	21.83333333	65
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	34083	21	64
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	34439	21.33333333	64
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	39939	22	66
1	1	1	1	1	0	1	1	1	1	1	1	1	1	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	0	27208	19.83333333	59
1	1	1	1	1	0	1	1	1	0	1	1	1	1	0	1	1	0	1	0	1	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	0	26208	19.5	58
1	0	1	0	1	0	0	0	1	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	3248	5.666666667	16
1	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1948	4.333333333	12
1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1448	3.333333333	9
1	0	1	1	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	2348	4.666666667	13
1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	400	1	3
1	1	1	1	1	0	1	1	1	0	1	0	1	1	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	0	18028	16	48
1	1	1	1	1	1	1	1	1	0	1	1	1	1	0	1	1	0	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	28453	20	60
1	1	1	1	1	0	1	0	1	0	1	0	1	1	0	1	1	0	0	0	0	0	0	1	0	1	1	0	1	1	0	0	0	1	1	1	1	0	1	1	1	0	13868	13.83333333	42
1	1	1	1	1	0	1	1	1	0	1	1	1	1	0	1	1	0	1	0	1	0	1	1	1	1	0	1	1	1	0	1	1	1	0	1	1	0	1	1	0	0	22528	18	54
1	1	1	1	1	0	0	0	1	0	1	1	1	1	0	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	27908	19.83333333	58

Figura 4.7: Frente de Pareto

Con el fin de visualizar mejor los resultados óptimos obtenidos por todos los algoritmos genéticos implementados en MULTIGEN, a continuación se presentan los gráficos tridimensionales que muestran los resultados de la optimización multi-objetivo.

Los Frentes de Pareto muestran la relación Reutilización-Compatibilidad-Costo en la configuración de características de una MPL utilizando los siguientes algoritmos: NSGA II (Figura 4.8), NSGA II SBX Classique (Figura 4.9), NSGA II b (Figura 4.10), NSGA II Mixte Continu-Entier (Figura 4.11), NSGA II Mixte Continu-Entier-Booléen (Figura 4.12), NSGA II Mixte Continu-Entier-Booléen LBCE (Figura 4.13) y MIB MOGA Continu-Entier-Booléen (Figura 4.14).

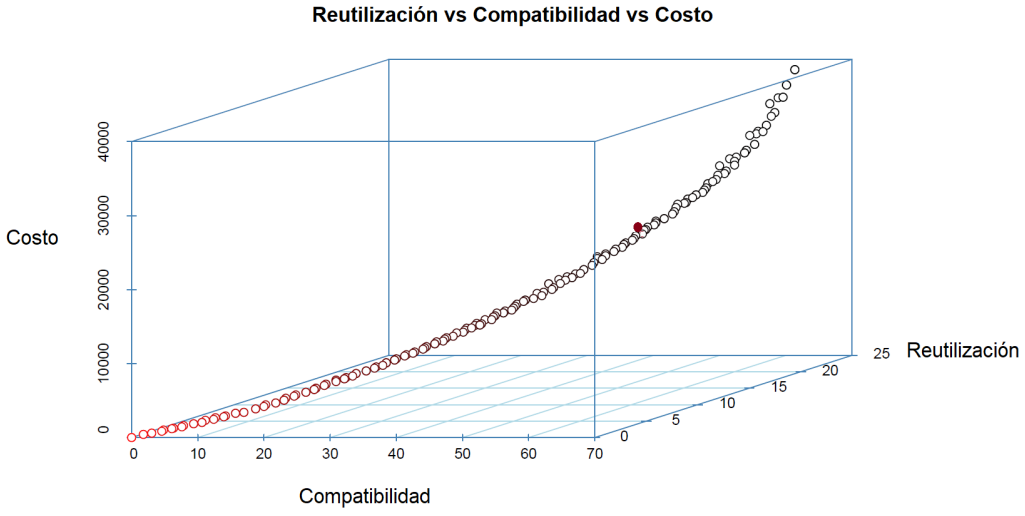


Figura 4.8: Frente de Pareto obtenido con el algoritmo NSGA II

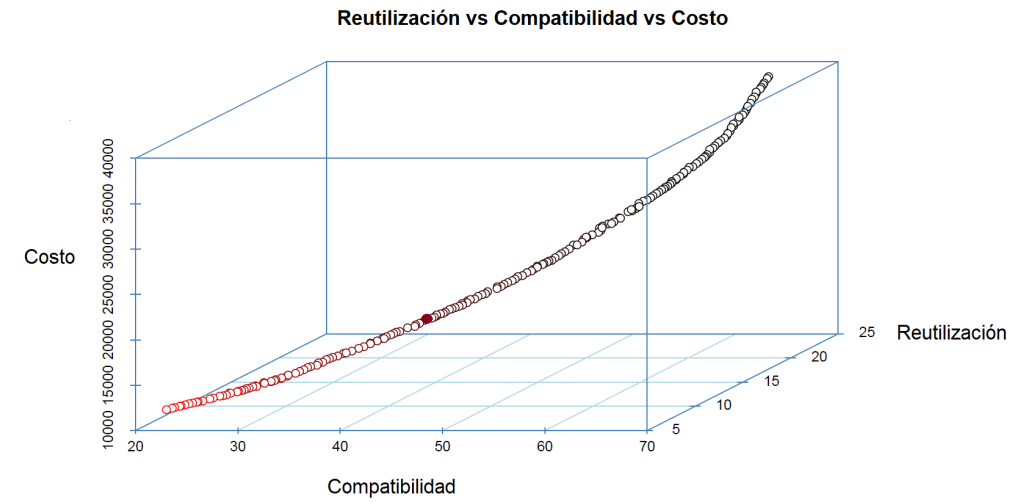


Figura 4.9: Frente de Pareto obtenido con el algoritmo NSGA II SBX Classique

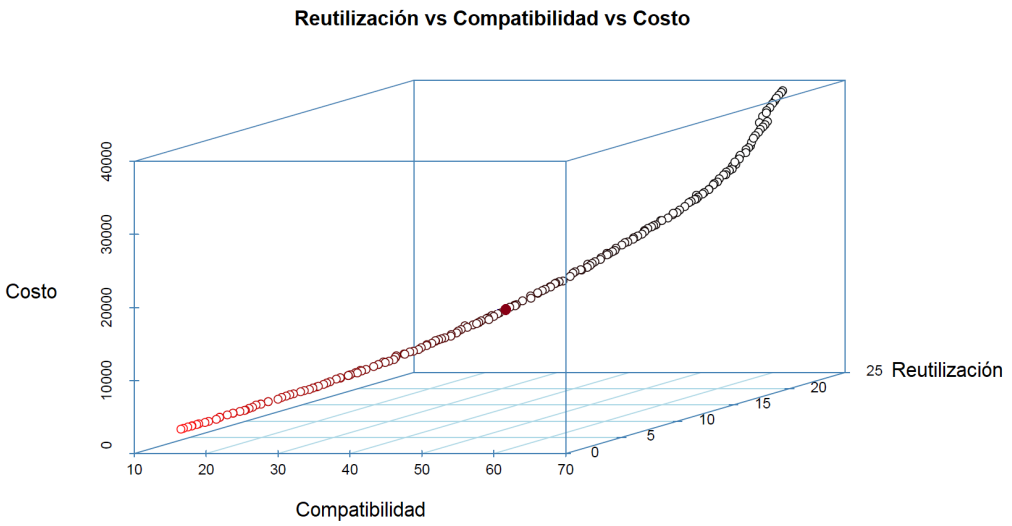


Figura 4.10: Frente de Pareto obtenido con el algoritmo NSGA II b

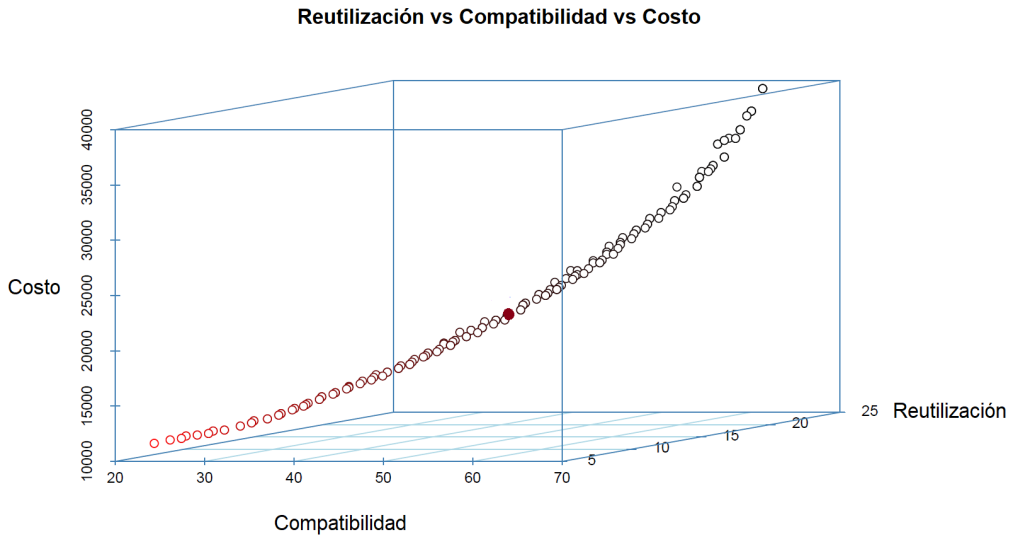


Figura 4.11: Frente de Pareto obtenido con el algoritmo NSGA II Mixte Continu-Entier

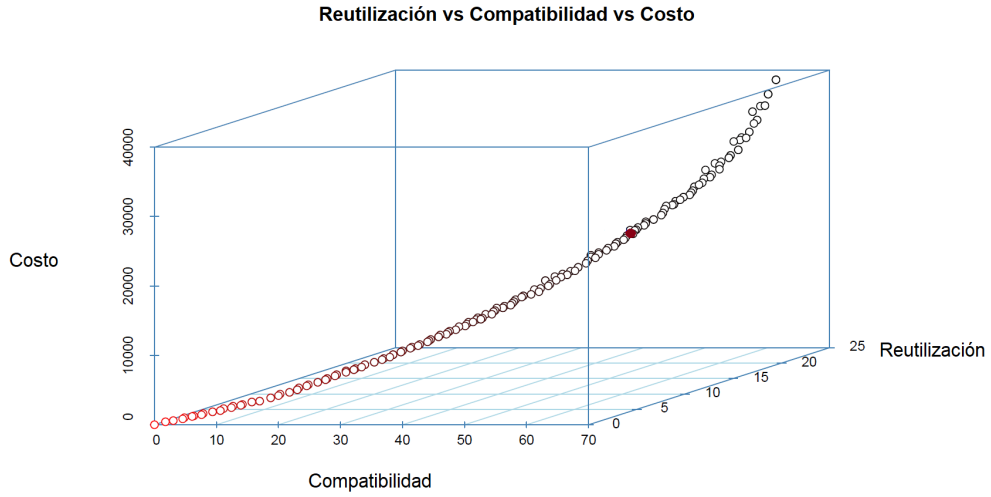


Figura 4.12: Frente de Pareto obtenido con el algoritmo NSGA II Mixte Continu-Entier-Booléen

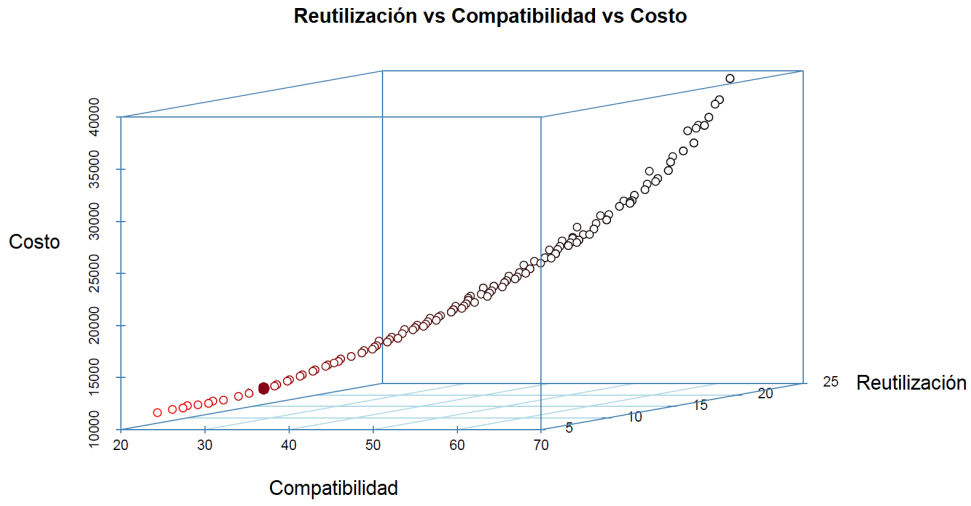


Figura 4.13: Frente de Pareto obtenido con el algoritmo NSGA II Mixte Continu-Entier-Booléen LBCE

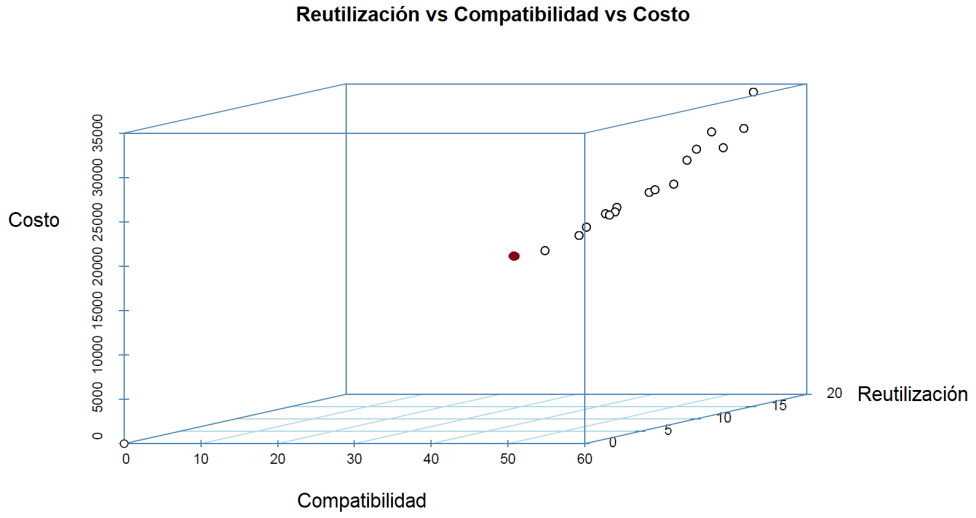


Figura 4.14: Frente de Pareto obtenido con el algoritmo MIB MOGA Continu-Entier-Booléen (LBCE)

4.4.4.2. Discusión de los resultados de optimización

Si bien existen herramientas computacionales para resolver problemas combinatorios y de optimización, estas pierden eficiencia conforme el número de variables incrementa, requiriendo

mayor tiempo y recurso computacional. Por otro lado, actualmente existen herramientas comerciales como CPLEX que resuelve problemas similares de una forma exacta o mediante metaheurísticos como RiskOptimizer. Sin embargo, estas herramientas tienen un costo significativo para su uso y la optimización que realizan es de forma mono-objetivo. Por esta razón, se han desarrollado diversos métodos metaheurísticos para tener alternativas más rápidas y alcanzables para este tipo de problemas (NP y multi-objetivo) como por ejemplo los algoritmos genéticos. Por otro lado, se identificó que no existe una solución exacta para el problema de la selección de características en una MPL porque es un problema multi-objetivo y es posible obtener un conjunto de soluciones factibles.

El problema se resolvió de forma mono-objetivo mediante IBM ILOG CPLEX Optimization Studio Community Edition obteniendo una solución exacta (Figura 4.15). La versión más reciente menciona que resuelve problemas de optimización multi-objetivo, sin embargo, esto no es posible porque genera una excepción (Figura 4.16).

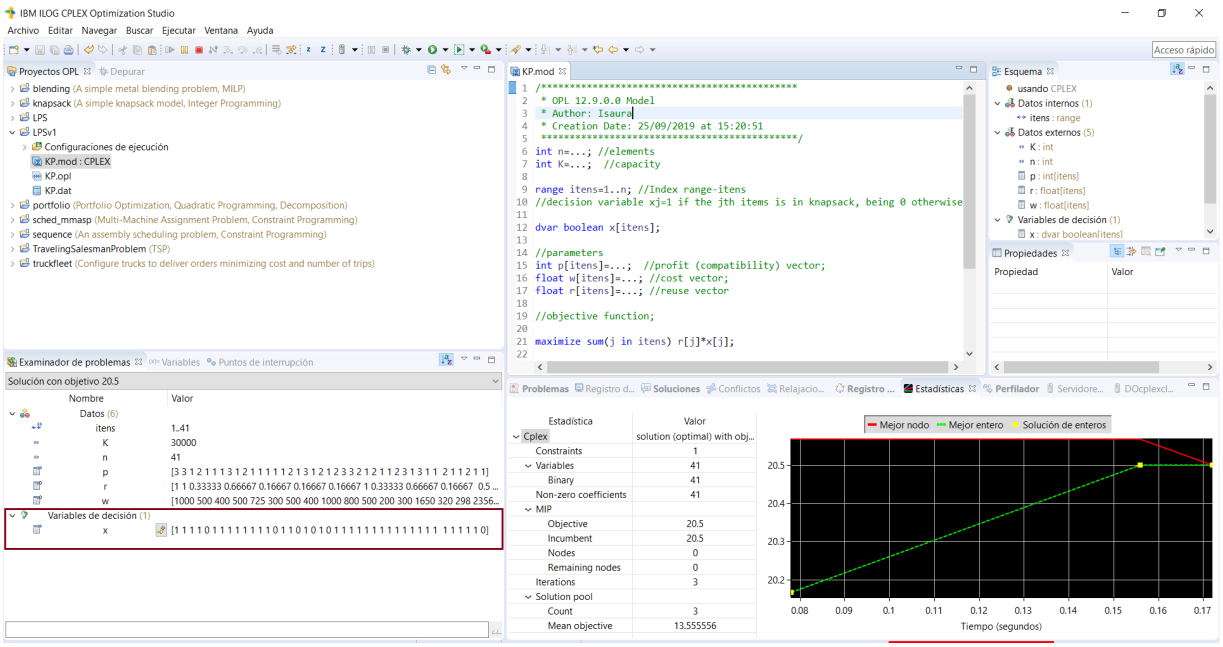


Figura 4.15: Resultados de la resolución del modelo de optimización utilizando CPLEX

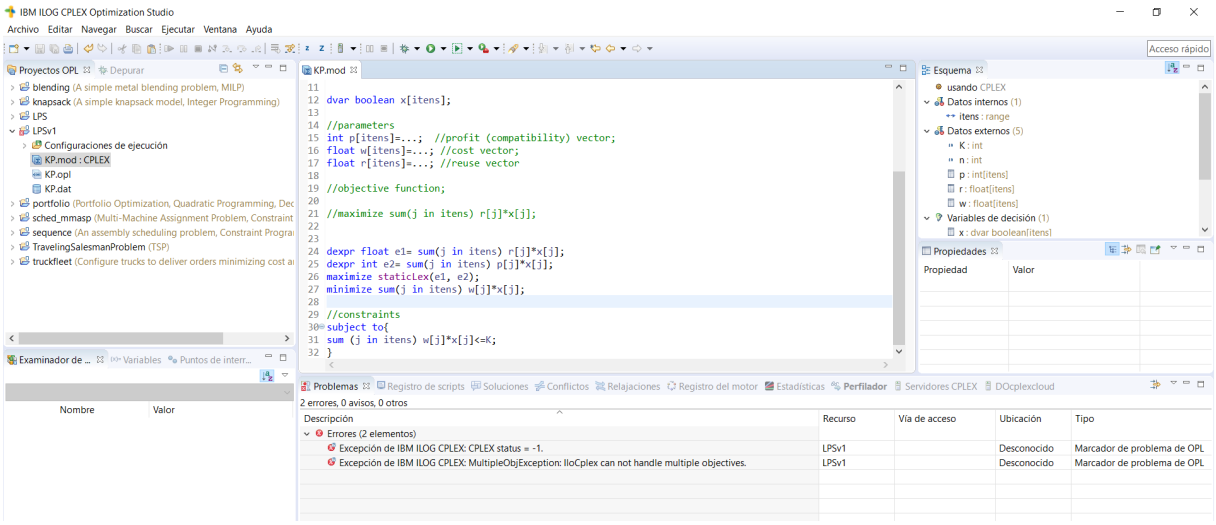


Figura 4.16: Excepción CPLEX

Para la generación de los productos de software, se consideró el escenario en el que el equilibrio entre los criterios técnicos (reutilización, compatibilidad) y el criterio económico (costo) tienen la misma importancia al elegir posibles configuraciones de características.

Después de analizar los resultados de ejecución de todas las variantes de los algoritmos NSGA II y MOGA (Figura 4.17), se determinó que la mejor alternativa al problema de optimización propuesto es la implementación de NSGA II Mixte Continu-Entier-Booléen, tanto en tiempo de ejecución (1060 segundos) como en los resultados obtenidos (Tabla 4.9).

En las pruebas, la configuración obtenida cumplió con las condiciones esperadas (Sección 4.4.4.1) y se obtuvo un costo de desarrollo de $20,728 con valores superiores a la reutilización esperada (17.33) y compatibilidad (51).

Tabla 4.9: Resultados obtenidos con los algoritmos genéticos

Algoritmo	Costo	Reutilización	Compatibilidad	Tiempo de ejecución (seg)
NSGA II	12390.67	9.5276	27.2361	1073
NSGA II SBX Classique	18135.54	13.9791	41.6974	1380
NSGA II b	14101.18	13.7945	41.5040	1416

Continúa en la página siguiente

Tabla4.9 Resultados obtenidos con los algoritmos genéticos (continuación)

Algoritmo	Costo	Reusabilidad	Compatibilidad	Tiempo de ejecución (seg)
NSGA II Mixte Continu-Entier	21618	16.3333	48	1353
NSGA II Mixte Continu-Entier-Booléen	20728	17.3333	51	1060
NSGA II Mixte Continu-Entier-Booléen LBCE	13098	10.5	30	1016
MIB MOGA Continu-Entier-Booléen	18325	12.3333	37	2137

La Tabla 4.10 muestra las configuraciones óptimas obtenidas por cada uno de los algoritmos genéticos. La configuración de productos en una MPL utilizando los resultados obtenidos brindan grandes beneficios a los desarrolladores ya que permite evaluar y comparar escenarios alternativos de reutilización en Líneas de Productos de Software y obtener un mayor beneficio económico ajustando la selección de características al costo de producción.

Tabla 4.10: Resultados obtenidos

NSGA II	NSGA II SBX Classique	NSGA II b	NSGA II Mixte Continu-Entier	NSGA II Mixte Continu-Entier-Booléen	NSGA II Mixte Continu-Entier-Booléen LBCE	MIB MOGAContinu-Entier-Booléen
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	0
1	1	1	1	1	1	0

Continúa en la página siguiente

Tabla4.10 Resultados obtenidos (continuación)

NSGA II	NSGA II SBX Classique	NSGA II b	NSGA II Mixte Continu-Entier	NSGA II Mixte Continu-Entier-Booléen	NSGA II Mixte Continu-Entier-Booléen LBCE	MIB MOGA Continu-Entier-Booléen
0	0	0	0	0	0	1
0	0	0	0	1	0	1
0	0	0	0	0	0	0
1	1	1	1	1	1	1
0	0	0	0	0	0	1
1	1	1	1	1	0	1
0	0	0	0	0	0	0
1	1	1	1	1	1	1
1	1	1	1	1	0	1
0	0	0	0	0	0	0
1	1	1	1	1	0	1
1	1	1	1	1	1	1
0	0	0	0	0	0	1
0	0	0	1	1	1	1
0	0	0	1	0	1	1
0	0	0	1	0	1	1
0	0	0	1	0	1	1
1	1	1	1	1	0	1
0	0	0	0	1	0	1
1	1	1	1	1	1	1
1	1	1	1	1	0	1
0	0	0	0	0	0	1
1	1	1	1	1	0	0
0	0	0	0	1	0	0
0	0	0	0	1	0	1
0	0	0	0	0	0	0
1	1	1	1	1	1	1
1	1	1	1	1	0	1
1	1	1	1	1	0	1

Continúa en la página siguiente

Tabla4.10 Resultados obtenidos (continuación)

NSGA II	NSGA II SBX Classique	NSGA II b	NSGA II Mixte Continu-Entier	NSGA II Mixte Continu-Entier-Booléen	NSGA II Mixte Continu-Entier-Booléen LBCE	MIB MOGA Continu-Entier-Booléen
0	0	0	0	0	0	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
0	0	0	1	1	0	1
1	1	1	1	1	0	0
1	1	1	1	1	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	1

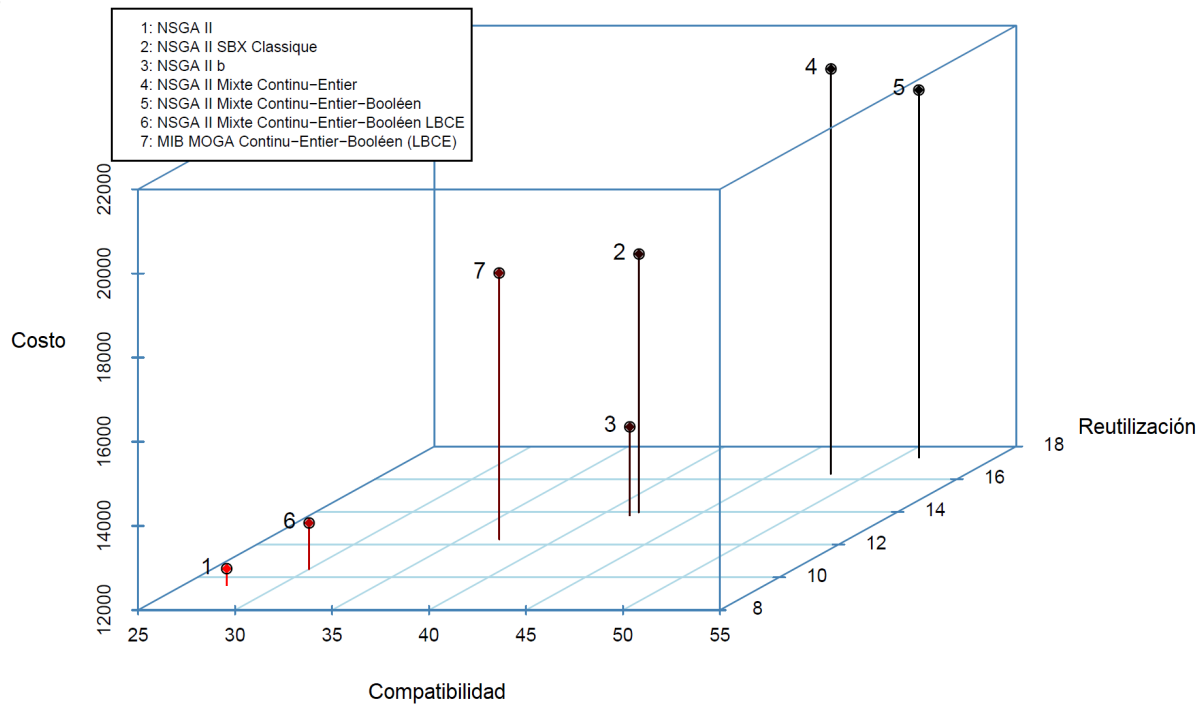


Figura 4.17: Resultados de la resolución del modelo de optimización utilizando algoritmos genéticos

Capítulo 5

Conclusiones

Hoy en día, las organizaciones necesitan reposicionar y rediseñar continuamente sus productos existentes o introducir nuevos productos en el mercado para mantener o mejorar su nivel de rentabilidad. Por otro lado, los clientes requieren productos de mejor calidad, precio competitivo y entrega más corta. Para lograr esto, es necesario cambiar la forma en que los productos son diseñados, fabricados y entregados. En este sentido, se detectan diversas oportunidades para el desarrollo de software de forma similar a las líneas de producción industrial. Actualmente, el proceso de diseño de nuevos productos de software se ha convertido en una tarea compleja de toma de decisiones que se encuentra influenciada por múltiples factores entre los que destacan varios tipos de relaciones de variabilidad y restricciones entre características, las configuraciones crecen exponencialmente de acuerdo con el número de características, lo que resulta en una explosión combinatoria de variantes, las características influyen de forma positiva o negativa en el producto, por mencionar algunos.

Las Líneas de Productos de Software utilizan un esquema de reutilización sistemática de activos de desarrollo en un dominio determinado. A pesar de su gran utilidad, los cambios constantes demandan la búsqueda de nuevos enfoques, técnicas o métodos como las Multilíneas de Productos de Software que apoyan el desarrollo de sistemas más complejos y brindan el soporte para la evolución y mantenimiento de las LPS sin poner en riesgo su funcionamiento. A través de la revisión del estado actual de los enfoques, herramientas y lenguajes que apoyan a las MPL,

se observa que a pesar de la experiencia que se tiene en la industria, la Ingeniería del Software requiere atender ciertos desafíos que se presentan al aumentar la complejidad de los sistemas, adicionar productos, actualizaciones por tecnología o cambio de requisitos, todo esto con el fin de satisfacer y cumplir las necesidades o expectativas del mercado. Asimismo, se detectan diversas áreas de oportunidad, especialmente en la fase de análisis de los modelos de características de las LPS, diseño de la MPL y pruebas.

La problemática de la configuración de productos de software se abordó bajo un enfoque de optimización multi-objetivo compensando el antagonismo entre los dos criterios técnicos (compatibilidad y reutilización) y el criterio económico, facilitando así la toma de decisiones al momento de elegir las posibles configuraciones. Además, el modelo garantiza la satisfacción de las restricciones del modelo de características de la MPL durante el proceso de configuración del producto.

Para administrar la variabilidad de múltiples Líneas de Productos de Software (MPL) y generar una nueva cartera de productos se propusó un método el cual ofrece una alternativa para desarrollar o mantener software personalizado que satisfaga las necesidades cambiantes de los clientes a través de técnicas de reutilización, separación de asuntos y de Ingeniería de Software Basada en Búsqueda. De esta forma se obtuvo una alternativa para desarrollar un modelo de características universal para una MPL (MPL-Feature Model) el cual surge por la necesidad de combinar los modelos de características de las LPS individuales con una técnica de Ingeniería de Software Basada en Búsqueda (SBSE).

Para ayudar al ingeniero de LPS a tomar decisiones específicamente en la Ingeniería de Aplicaciones durante el proceso de configuración de productos (selección de características) en una MPL se propusó un modelo de optimización multi-objetivo que facilita la selección y combinación de características proporcionadas por múltiples LPS ya que reduce la carga mental y complejidad cognitiva asociadas con el proceso de configuración.

Respecto a la toma de decisiones durante la fase de configuración de la MPL, las técnicas de Ingeniería de Software Basada en Búsqueda como los algoritmos genéticos ofrecen una alternativa viable para combinar los insumos de las LPS ya que visualizar las diferentes combinaciones válidas de productos es una tarea compleja y propensa a errores.

La solución del modelo de optimización mediante algoritmos genéticos permite obtener productos de software que mejor se adapten a los requisitos de las partes interesadas maximizando la reutilización y compatibilidad entre las características y minimizando el costo de desarrollo de los productos de software. Los resultados obtenidos con el modelo facilitan el diseño del modelo de características de la MPL y determinar así el alcance de la MPL.

Para guiar el diseño de arquitecturas dentro del dominio de la automatización se obtuvo una Arquitectura de Referencia de la MPL Inmótica la cual muestra las partes comunes y variables de las LPS para el desarrollo incremental de productos

Para construir aplicaciones en el dominio de la automatización se obtuvo una plataforma para la MPL, es decir se obtuvieron una serie de artefactos reutilizables que facilitan el desarrollo de aplicaciones que controlan entornos físicos equipados con tecnología heterogénea como viviendas, oficinas, hoteles, invernaderos, entre otros específicamente en los campos de la Domótica, Inmótica e Internet de las Cosas (IoT). Asimismo, se obtuvo una Multilínea de Productos de Software (MPL) como una forma para obtener y/o complementar productos de software a partir de la combinación y reutilización de los insumos de las LPS disponibles.

5.1. Trabajo a futuro

Como trabajo a futuro se pretende actualizar e introducir nuevas funcionalidades a la MPL: refinar y complementar el modelo de optimización multi-objetivo propuesto con variables (características) adicionales para personalizar otros productos de software y abarcar otros segmentos de mercado relacionados a la automatización inteligente de edificios (interiores y exteriores) y así tener un mayor variabilidad de productos de software.

Como trabajo adicional, se vislumbra utilizar el caso de estudio y resolver el modelo de optimización con otras técnicas de Ingeniería de Software Basada en Búsqueda (por ejemplo, colonia de hormigas o recocido simulado). Asimismo, el modelo podría utilizarse como base para aplicarse en otras etapas del ciclo de vida (pruebas, ubicación de características, mantenimiento) de las MPL.

Apéndice A

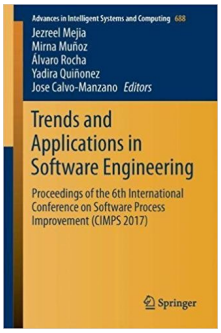
Productos académicos

A continuación, se describen los productos académicos derivados de este proyecto de investigación:

Artículo JCR

Trujillo-Tzanahua, Guadalupe-Isaura; Juarez-Martinez, Ulises; Aguilar-Lasserre, Alberto-Alfonso; Cortes-Verdin, Maria-Karen; Azzaro-Pantel, Catherine: 'Multiple Software Product Lines to configure applications of Internet of Things', IET Software, 2019, DOI: 10.1049/iet-sen.2019.0032
IET Digital Library,
https://digital-library.theiet.org/content/journals/10.1049/iet-sen.2019.0032

Conferencia Internacional



Trujillo-Tzanahua G.I., Juárez-Martínez U., Aguilar-Lasserre A.A., Cortés-Verdín M.K. (2018) Multiple Software Product Lines: applications and challenges. In: Mejia J., Muñoz M., Rocha Á., Quiñonez Y., Calvo-Manzano J. (eds) Trends and Applications in Software Engineering. CIMPS 2017. Advances in Intelligent Systems and Computing, vol 688. Springer, Cham
ISBN: 978-3-319-69341-5
DOI: https://doi.org/10.1007/978-3-319-69341-5_11

Apéndice B

Frente de Pareto

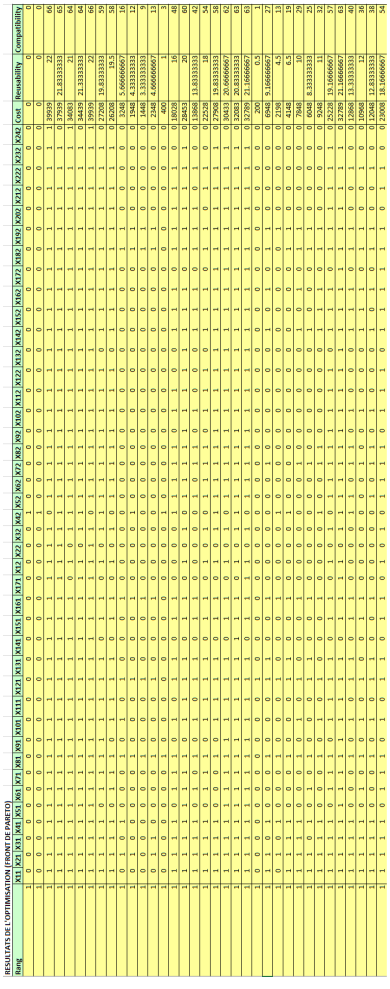


Figura B.1: Frente de Pareto

Referencias

[1] E. Y. Nakagawa and F. Oquendo, "Perspectives and challenges of reference architectures in multi software product line," in *Proceedings of the 17th International Software Product Line Conference Co-located Workshops*. New York, NY, USA: ACM, 2013, pp. 100–103.

[2] P. C. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*. Boston, MA, USA: Addison-Wesley Professional, 2001.

[3] K. Pohl, G. Böckle, and F. J. v. d. Linden, *Software Product Line Engineering: Foundations, Principles aand Techniques*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[4] M. A. Ramos, P. C. Masiero, R. T. V. Braga, and R. A. D. Penteado, "From software product lines to system of systems: Analysis of an evolution path," in *Information Reuse and Integration (IRI), 2013 IEEE 14th International Conference on*. IEEE, August 2013, pp. 394–401.

[5] J. Bosch, "From software product lines to software ecosystems," in *Proceedings of the 13th International Software Product Line Conference*. Pittsburgh, PA, USA: Carnegie Mellon University, 2009, pp. 111–119.

[6] C. W. Krueger, "New methods in software product line development," in *10th International Software Product Line Conference (SPLC'06)*. IEEE, 2006, pp. 95–99.

[7] F. G. Marinho, R. M. C. Andrade, C. Werner, W. Viana, M. E. F. Maia, L. S. Rocha, E. Teixeira, J. a. B. F. Filho, V. L. L. Dantas, F. Lima, and S. Aguiar, "Mobiline: A nested software product line for the domain of mobile and context-aware applications," *Science of Computer Programming*, vol. 78, no. 12, pp. 2381 – 2398, 2013.

[8] M. Acher, P. Collet, P. Lahire, and R. France, "Managing multiple software product lines using merging techniques," University of Nice Sophia Antipolis, Tech. Rep., 2010.

[9] M. Acher, P. Collet, P. Lahire, and R. B. France, "A domain-specific language for managing feature models," in *Proceedings of the 2011 ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2011, pp. 1333–1340.

[10] G. Holl, P. Grünbacher, and R. Rabiser, "A systematic review and an expert survey on capabilities supporting multi product lines," *Information and Software Technology*, vol. 54, no. 8, pp. 828–852, 2012.

[11] M. Lienhardt, F. Damiani, S. Donetti, and L. Paolini, "Multi software product lines in the wild," in *Proceedings of the 12th International Workshop on Variability Modelling of Software-Intensive Systems - VAMOS 2018*. ACM, 2018, pp. 89–96.

[12] R. Rabiser, P. Grünbacher, and G. Holl, "Improving awareness during product derivation in multi-user multi product line environments," in *Proceedings 1st Int'l Workshop on Automated Configuration and Tailoring of Applications (ACoTA 2010) in conjunction with 25th IEEE/ACM Int'l Conference on Automated Software Engineering (ASE 2010), Antwerp, Belgium, September*, 2010, pp. 1–5.

[13] R. van Ommering, *Beyond Product Families: Building a Product Population?* Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 187–198.

[14] M. Rosenmüller and N. Siegmund, "Automating the configuration of multi software product lines," in *Proceedings of Fourth International Workshop on Variability Modelling of Software-Intensive Systems*, 2010, pp. 123–130.

[15] J. Joshua, D. Alao, S. Okolie, and O. Awodele, "Software ecosystem: Features, benefits and challenges," *International Journal of Advanced Computer Science and Applications(IJACSA)*, vol. 4, no. 8, 2013.

[16] C. Seidl and U. ABssmann, "Towards modeling and analyzing variability in evolving software ecosystems," in *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*. New York, NY, USA: ACM, 2013, pp. 3:1–3:8.

[17] S. Urli, M. Blay-Fornarino, P. Collet, S. Mosser, and M. Riveill, "Managing a software ecosystem using a multiple software product line: a case study on digital signage systems," in *Euromicro Conference series on Software Engineering and Advanced Applications(SEAA'14)*, August 2014, pp. 344–351.

[18] L. Teixeira, P. Borba, and R. Gheyi, "Safe evolution of product populations and multi product lines," in *Proceedings of the 19th International Conference on Software Product Line*. New York, NY, USA: ACM, 2015, pp. 171–175.

[19] G. Holl, D. Thaller, P. Grünbacher, and C. Elsner, "Managing emerging configuration dependencies in multi product lines," in *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*. New York, NY, USA: ACM, 2012, pp. 3–10.

[20] G. Holl, C. Elsner, P. Grünbacher, and M. Vierhauser, "An infrastructure for the life cycle management of multi product lines," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2013, pp. 1742–1749.

[21] T. Klambauer, G. Holl, and P. Grünbacher, "Monitoring system-of-systems requirements in multi product lines," in *Proceedings of the 19th International Conference on Requirements Engineering: Foundation for Software Quality*. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 379–385.

[22] J. Savolainen, T. Männistö, and V. Myllärniemi, *Experiences in System-of-Systems-Wide Architecture Evaluation over Multiple Product Lines*. Cham: Springer International Publishing, 2014, ch. Software Reuse for Dynamic Systems in the Cloud and Beyond, pp. 58–72.

[23] K. Schmid and H. Eichelberger, "Easy-producer: From product lines to variability-rich software ecosystems," in *Proceedings of the 19th International Conference on Software Product Line*. New York, NY, USA: ACM, 2015, pp. 390–391.

[24] H. Eichelberger and K. Schmid, "Ivml: A dsl for configuration in variability-rich software ecosystems," in *Proceedings of the 19th International Conference on Software Product Line*. New York, NY, USA: ACM, 2015, pp. 365–369.

[25] K. Schmid, "Variability support for variability-rich software ecosystems," in *Product Line Approaches in Software Engineering (PLEASE), 2013 4th International Workshop on*, May 2013, pp. 5–8.

[26] C. B. Nielsen, P. G. Larsen, J. Fitzgerald, J. Woodcock, and J. Peleska, "Systems of systems engineering: Basic concepts, model-based techniques, and research directions," *ACM Comput. Surv*, vol. 48, no. 2, pp. 18:1–18:41, September 2015.

[27] K. Czarnecki and U. W. Eisenecker, *Generative Programming: Methods, Tools, and Applications*. ACM Press/Addison-Wesley Publishing Co., 2000.

[28] K. Czarnecki, "Overview of generative software development," in *Unconventional Programming Paradigms: International Workshop UPP 2004, Le Mont Saint Michel, France, September 15-17, 2004, Revised Selected and Invited Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 326–341.

[29] ——, "Generative programming: Principles and techniques of software engineering based on automated configuration and fragment-based component models," Ph.D. dissertation, Technische Universitat Ilmenau, Germany, 1999.

[30] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin, *Aspect-oriented programming*. Springer Berlin Heidelberg, 1997, ch. Proceedings of the European Conference on Object-Oriented Programming (ECOOP), pp. 220–242.

[31] K. Schmid, "A comprehensive product line scoping approach and its validation," in *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, 2002, pp. 593–603.

[32] F. Bachmann, L. Bass, P. Clements, D. Garlan, J. Ivers, M. Little, P. Merson, R. Nord, and J. Stafford, *Documenting software architectures: views and beyond*, 2nd ed. Addison-Wesley Professional, 2010.

[33] M. Rosenmüller, N. Siegmund, T. Thüm, and G. Saake, "Multi-dimensional variability modeling," in *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*. New York, NY, USA: ACM, 2011, pp. 11–20.

[34] S. Deelstra, M. Sinnema, and J. Bosch, "Product derivation in software product families: a case study," *Journal of Systems and Software*, vol. 74, no. 2, pp. 173 – 194, 2005, the new context for software engineering education and training.

[35] D. Benavides, P. Trinidad, and A. Ruiz-Cortés, "Automated reasoning on feature models," in *Proceedings of the 17th International Conference on Advanced Information Systems Engineering*, 2005, pp. 491–503.

[36] D. Benavides, S. Segura, and A. Ruiz-Cortés, "Automated analysis of feature models 20 years later: A literature review," *Information Systems*, vol. 35, no. 6, pp. 615–636, 2010.

[37] U. Afzal, T. Mahmood, and Z. Shaikh, "Intelligent software product line configurations: A literature review," *Computer Standards & Interfaces*, vol. 48, pp. 30–48, 2016.

[38] M. Aoyama, "Continuous and discontinuous software evolution: Aspects of software evolution across multiple product lines," in *Proceedings of the 4th International Workshop on Principles of Software Evolution*. New York, NY, USA: ACM, 2001, pp. 87–90.

[39] M. Aoyama, K. Watanabe, Y. Nishio, and M. Yasuyuki, "Embracing requirements variety for e-governments based on multiple product-lines frameworks," in *Requirements Engineering Conference, 2003. Proceedings. 11th IEEE International*. IEEE, September 2003, p. 285.

[40] S. Bühne, K. Lauenroth, and K. Pohl, "Why is it not sufficient to model requirements variability with feature models?" Proceedings of the automotive requirements engineering AURE04, 2004.

[41] M. Acher, P. Collet, A. Gaignard, P. Lahire, J. Montagnat, and R. B. France, "Composing multiple variability artifacts to assemble coherent workflows," *Software Quality Journal*, vol. 20, no. 3, pp. 689–734, 2012.

[42] M. R. A. Setyautami, D. Adianto, and A. Azurat, "Modeling multi software product lines using uml," in *Proceeedings of the 22nd International Conference on Systems and Software Product Line - SPLC '18*. New York, New York, USA: ACM Press, 2018, pp. 274–278.

[43] F. Damiani, M. Lienhardt, and L. Paolini, "A formal model for multi software product lines," *Science of Computer Programming*, vol. 172, pp. 203–231, mar 2019.

[44] B. Tekinerdogan, Özgü Özköse Erdogan, and O. Aktug, "Archample - architectural analysis approach for multiple product line engineering," in *Relating System Quality and Software Architecture*, I. Mistrik, R. Bahsoon, P. Eeles, R. Roshandel, and M. Stal, Eds. Boston: Morgan Kaufmann, 2014, ch. 10, pp. 263–285.

[45] B. Tekinerdogan, E. Cilden, O. O. Erdogan, and O. Aktug, "Architecture conformance analysis approach within the context of multiple product line engineering," in *Proceedings of the 2014 23rd Australian Software Engineering Conference*. Washington, DC, USA: IEEE Computer Society, April 2014, pp. 25–28.

[46] H. Hartmann, T. Trew, and A. Matsinger, "Supplier independent feature modelling," in *Proceedings of the 13th International Software Product Line Conference*. Pittsburgh, PA, USA: Carnegie Mellon University, 2009, pp. 191–200.

[47] J. Savolainen, M. Mannion, and J. Kuusela, "Developing platforms for multiple software product lines," in *Proceedings of the 16th International Software Product Line Conference - Volume 1*. New York, NY, USA: ACM, 2012, pp. 220–228.

[48] D. Dhungana, D. Seichter, G. Botterweck, R. Rabiser, P. Grünbacher, D. Benavides, and J. A. Galindo, "Configuration of multi product lines by bridging heterogeneous variability modeling approaches," in *Proceedings of the 2011 15th International Software Product Line Conference*. Washington, DC, USA: IEEE Computer Society, 2011, pp. 120–129.

[49] ——, "Integrating heterogeneous variability modeling approaches with invar," in *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*. New York, NY, USA: ACM, 2013, pp. 8:1–8:5.

[50] J. A. Galindo, D. Dhungana, R. Rabiser, D. Benavides, G. Botterweck, and P. Grünbacher, "Supporting distributed product configuration by integrating heterogeneous variability modeling approaches," *Information and Software Technology*, vol. 62, pp. 78–100, 2015.

[51] F. V. D. Linden and J. G. Wijnstra, "Platform engineering for the medical domain," *4th International Workshop on Software Product-Family Engineering*, pp. 224–237, 2002.

[52] C. Brink, M. Peters, and S. Sachweh, "Configuration of mechatronic multi product lines," in *Proceedings of the 3rd International Workshop on Variability & Composition*. New York, NY, USA: ACM, 2012, pp. 7–12.

[53] S. El-Sharkawy, C. Kröher, and K. Schmid, "Supporting heterogeneous compositional multi software product lines," in *Proceedings of the 15th International Software Product Line Conference, Volume 2*. New York, NY, USA: ACM, 2011, pp. 25:1–25:4.

[54] S. Urli, M. Blay-Fornarino, and P. Collet, "Handling complex configurations in software product lines: a tooled approach," in *Proceedings of the 18th International Software Product Line Conference - Volume 1,*. New York, NY, USA: ACM, 2014, pp. 112–121.

[55] A. Rahmat, S. Kassim, M. H. Selamat, and S. Hassan, "Actor in multi product line," in *Proceedings of the 10th International Conference on Ubiquitous Information Management and Communication*. New York, NY, USA: ACM, 2016, pp. 61:1–61:8.

[56] M. Rosenmüller, N. Siegmund, C. Kästner, and S. S. u. Rahman, "Modeling dependent software product lines," in *Proceedings of the GPCE Workshop on Modularization, Composition and Generative Techniques for Product Line Engineering (McGPLE)*. Department of Informatics and Mathematics, University of Passau, October 2008, pp. 13–18.

[57] G. Holl, "Product line bundles to support product derivation in multi product lines," in *Proceedings of the 15th International Software Product Line Conference, Volume 2*. New York, NY, USA: ACM, 2011, pp. 41:1–41:6.

[58] R. Schröter, T. Thüm, N. Siegmund, and G. Saake, "Automated analysis of dependent feature models," in *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*. New York, NY, USA: ACM, 2013, pp. 9:1–9:5.

[59] R. Schröter, "Using multi-level interfaces to improve analyses of multi product lines," University of Magdeburg, Germany, Tech. Rep. 4, October 2014.

[60] H. Hartmann and T. Trew, "Using feature diagrams with context variability to model multiple product lines for software supply chains," in *Proceedings of the 12th International Software Product Lines Conference*. Washington, DC, USA: IEEE Computer Society, September 2008, pp. 12–21.

[61] N. Khedri and R. Khosravi, "Towards managing data variability in multi product lines," in *Proceedings of the 3rd International Conference on Model-Driven Engineering and*

Software Development. Portugal: SCITEPRESS - Science and Technology Publications, Lda, February 2015, pp. 523–530.

[62] R. Schröter, N. Siegmund, and T. Thüm, "Towards modular analysis of multi product lines," in *Proceedings of the 17th International Software Product Line Conference Co-located Workshops*. New York, NY, USA: ACM, 2013, pp. 96–99.

[63] B. Tekinerdogan, O. O. Erdogan, and O. Aktug, "Supporting incremental product development using multiple product line architecture," *International Journal of Knowledge and Systems Science (IJKSS)*, vol. 5, no. 4, pp. 1–16, October 2014.

[64] C. Brink, P. Heisig, and F. Wackermann, *Change Impact in Product Lines: A Systematic Mapping Study*. Cham: Springer International Publishing, 2016, pp. 677–694.

[65] N. Dintzner, U. Kulesza, A. van Deursen, and M. Pinzger, *Evaluating Feature Change Impact on Multi-product Line Configurations Using Partial Information*. Springer International Publishing, 2014, pp. 1–16.

[66] M. Mendonca, D. Cowan, and T. Oliveira, "A process-centric approach for coordinating product configuration decisions," in *Proceedings of the 40th Annual Hawaii International Conference on System Sciences*. IEEE, January 2007, pp. 283a–.

[67] R. Rabiser, D. Dhungana, P. Grunbacher, K. Lehner, and C. Federspiel, "Involving non-technicians in product derivation and requirements engineering: A tool suite for product line engineering," in *15th IEEE International Requirements Engineering Conference (RE 2007)*, October 2007, pp. 367–368.

[68] C. Elsner, D. Lohmann, and W. Schröder-Preikschat, "Product derivation for solution-driven product line engineering," in *Proceedings of the First International Workshop on Feature-Oriented Software Development*. New York, NY, USA: ACM, 2009, pp. 35–41.

[69] M. Harman, "The role of artificial intelligence in software engineering," in *2012 1st International Workshop on Realizing AI Synergies in Software Engineering, RAISE 2012 - Proceedings*. IEEE, june 2012, pp. 1–6.

[70] M. Harman and B. F. Jones, "Search-based software engineering," *Information and Software Technology*, vol. 43, no. 14, pp. 833–839, December 2001.

[71] M. Harman and J. Clark, "Metrics are fitness functions too," in *Proceedings - International Software Metrics Symposium*. IEEE, 2004, pp. 58–69.

[72] L. Tan, Y. Lin, H. Ye, and G. Zhang, "Improving product configuration in software product line engineering," in *Conferences in Research and Practice in Information Technology Series*, vol. 135, 2013, pp. 125–134.

[73] J. Guo, J. White, G. Wang, J. Li, and Y. Wang, "A genetic algorithm for optimized feature selection with resource constraints in software product lines," *Journal of Systems and Software*, vol. 84, no. 12, pp. 2208 – 2221, 2011.

[74] J. Cruz, P. S. Neto, R. Britto, R. Rabelo, W. Ayala, T. Soares, and M. Mota, "Toward a hybrid approach to generate software product line portfolios," in *2013 IEEE Congress on Evolutionary Computation*, June 2013, pp. 2229–2236.

[75] A. S. Sayyad, T. Menzies, and H. Ammar, "On the value of user preferences in search-based software engineering: A case study in software product lines," in *Proceedings of the 2013 International Conference on Software Engineering*. Piscataway, NJ, USA: IEEE Press, 2013, pp. 492–501.

[76] U. Afzal, T. Mahmood, and I. R. andZubair Ahmed Shaikh, "Minimizing feature model inconsistencies in software product lines," in *17th IEEE International Multi Topic Conference 2014*, December 2014, pp. 137–142.

[77] Z. Zhan, W. Luo, Z. Guo, and Y. Liu, "Feature selection optimization based on atomic set and genetic algorithm in software product line," in *Advances in Intelligent Systems and Computing*. Springer, Cham, 2018, vol. 686, pp. 93–100.

[78] Y.-l. Wang and J.-w. Pang, "Ant colony optimization for feature selection in software product lines," *Journal of Shanghai Jiaotong University (Science)*, vol. 19, no. 1, pp. 50–58, 2014.

[79] J. A. Pereira, P. Matuszyk, S. Krieter, M. Spiliopoulou, and G. Saake, "A feature-based personalized recommender system for product-line configuration," in *Proceedings of the 2016 ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences*. New York, NY, USA: ACM, 2016, pp. 120–131.

[80] E. L. Féderle, T. do Nascimento Ferreira, T. E. Colanzi, and S. R. Vergilio, "Opla-tool: A support tool for search-based product line architecture design," in *Proceedings of the 19th International Conference on Software Product Line*, ser. SPLC '15. New York, NY, USA: ACM, 2015, pp. 370–373.

[81] A. Z. Chohan, A. Bibi, and Y. Hafeez Motla, "Optimized software product line architecture and feature modeling in improvement of spl," *Proceedings - 2017 International Conference on Frontiers of Information Technology, FIT 2017*, pp. 167–172, Dec 2018.

[82] J. Font, L. Arcega, and Ø. H. Cetina, "Feature location in model-based software product lines through a genetic algorithm," in *Software Reuse: Bridging with Social-Awareness*. Springer International Publishing, 2016.

[83] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. L. Traon, "Multi-objective test generation for software product lines," in *Proceedings of the 17th International Software Product Line Conference*. New York, NY, USA: ACM, 2013, pp. 62–71.

[84] S. Wang, S. Ali, and A. Gotlieb, "Cost-effective test suite minimization in product lines using search techniques," in *Journal of Systems and Software*, vol. 103. Elsevier, may 2015, pp. 370–391.

[85] G. Perrouin, S. Oster, S. Sen, J. Klein, B. Baudry, and Y. le Traon, "Pairwise testing for software product lines: Comparison of two approaches," *Software Quality Journal*, vol. 20, no. 3-4, pp. 605–643, September 2012.

[86] Y. A. Alsariera, M. A. Majid, and K. Z. Zamli, "Splba: An interaction strategy for testing software product lines using the bat-inspired algorithm," in *2015 4th International Confe-*

rence on Software Engineering and Computer Systems, ICSECS 2015: Virtuous Software Solutions for Big Data. IEEE, aug 2015, pp. 148–153.

[87] H. I. Alsawalqah, S. Kang, and J. Lee, "A method to optimize the scope of a software product platform based on end-user features," *Journal of Systems and Software*, vol. 98, pp. 79–106, dec 2014.

[88] S. Robak and A. Pieczynski, "Employing fuzzy logic in feature diagrams to model variability in software product-lines," in *10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*, April 2003, pp. 305–311.

[89] T. E. Colanzi, S. R. Vergilio, I. M. S. Gimenes, and W. N. Oizumi, "A search-based approach for software product line design," in *Proceedings of the 18th International Software Product Line Conference on - SPLC '14*. New York, New York, USA: ACM Press, 2014, pp. 237–241.

[90] T. H. Tan, Y. Xue, M. Chen, J. Sun, Y. Liu, and J. S. Dong, "Optimizing selection of competing features via feedback-directed evolutionary algorithms," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*. New York, NY, USA: ACM, 2015, pp. 246–256.

[91] Y. Xue, J. Zhong, T. H. Tan, Y. Liu, W. Cai, M. Chen, and J. Sun, "Ibed: Combining ibea and de for optimal feature selection insoftware product line engineering," *Applied Soft Computing*, vol. 49, pp. 1215–1231, 2016.

[92] P. de Alcântara dos Santos Neto, R. Britto, R. de Andrade Lira Rabêlo, J. J. de Almeida Cruz, and W. A. L. Lira, "A hybrid approach to suggest software product line portfolios," *Applied Soft Computing*, vol. 49, pp. 1243–1255, 2016.

[93] M. Asadi, S. Soltani, D. Gasevic, M. Hatala, and E. Bagheri, "Toward automated feature model configuration with optimizing non-functional requirements," *Information and Software Technology*, vol. 56, no. 9, pp. 1144 – 1165, 2014.

[94] J. White, B. Dougherty, and D. C. Schmidt, "Selecting highly optimal architectural feature sets with filtered cartesian flattening," *Journal of Systems and Software*, vol. 82, no. 8, pp. 1268 – 1284, 2009.

[95] N. Siegmund, M. Rosenmüller, M. Kuhlemann, C. Kästner, S. Apel, and G. Saake, "Spl conqueror: Toward optimization of non-functional properties in software product lines," *Software Quality Journa*, vol. 20, no. 3, pp. 487–517, 2012.

[96] X. Lian and L. Zhang, "Optimized feature selection towards functional and non-functional requirements in software product lines," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, March 2015, pp. 191–200.

[97] R. Olaechea, D. Rayside, J. Guo, and K. Czarnecki, "Comparison of exact and approximate multi-objective optimization for software product lines," in *Proceedings of the 18th International Software Product Line Conference - Volume 1*. New York, NY, USA: ACM, 2014, pp. 92–101.

[98] D. Brevet, T. Saber, G. Botterweck, and A. Ventresque, "Preliminary study of multi-objective features selection for evolving software product lines," in *Search Based Software Engineering: 8th International Symposium, SSBSE 2016, Raleigh, NC, USA, October 8-10, 2016, Proceedings*. Springer International Publishing, 2016, pp. 274–280.

[99] E. Bagheri, M. Asadi, D. Gasevic, and S. Soltani, "Stratified analytic hierarchy process: Prioritization and selection of software feature," in *Proceedings of the 14th International Conference on Software Product Lines: Going Beyond*. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 300–315.

[100] A. S. Sayyad, J. Ingram, T. Menzies, and H. Ammar, "Optimum feature selection in software product lines: Let your model and values guide your search," in *Proceedings of the 1st International Workshop on Combining Modelling and Search-Based Software Engineering*. Piscataway, NJ, USA: IEEE Press, 2013, pp. 22–27.

[101] S. Chen and M. Erwig, "Optimizing the product derivation process," in *2011 15th International Software Product Line Conference*, August 2011, pp. 35–44.

[102] T. Thüm, T. Winkelmann, R. Schröter, M. Hentschel, and S. Krüger, "Variability hiding in contracts for dependent software product lines," in *Proceedings of the Tenth International Workshop on Variability Modelling of Software-intensive Systems*. New York, NY, USA: ACM, 2016, pp. 97–104.

[103] K. Czarnecki, S. Helsen, and U. Eisenecker, "Staged configuration through specialization and multilevel configuration of feature models," *Software Process: Improvement and Practice*, vol. 10, no. 2, pp. 143–169, 2005.

[104] R. van Ommering, "Building product populations with software components," in *Proceedings of the 24th International Conference on Software Engineering*. New York, NY, USA: ACM, 2002, pp. 255–265.

[105] R. Van Ommering, "Software reuse in product populations," *IEEE Transactions on Software Engineering*, vol. 31, no. 7, pp. 537–550, July 2005.

[106] S. El-Sharkawy, C. Kröher, and K. Schmid, "Support for complex product line populations," in *Proceedings of the 15th International Software Product Line Conference, Volume 2*. New York, NY, USA: ACM, 2011, pp. 47:1–47:1.

[107] J. M. Thompson and M. P. E. Heimdahl, "Structuring product family requirements for n-dimensional and hierarchical product lines," *Requirements Engineering*, vol. 8, no. 1, pp. 42–54, February 2003.

[108] E. Rommes and J. G. Wijnstra, "Implementing a reuse strategy: Architecture, process and organization aspects of a medical imaging product family," in *Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences - Volume 09*, January 2005, pp. 312a–312a.

[109] S. Trujillo, C. Kästner, and S. Apel, "Product lines that supply other product lines: A service-oriented approach," in *Proceedings of the SPLC Workshop on Service-Oriented*

Architectures and Product Lines (SOAPL). Pittsburgh, PA: SEI, September 2007, pp. 69–76.

[110] N. Altintas and S. Cetin, "Managing large scale reuse across multiple software product lines," in *Proceedings of the 10th International Conference on Software Reuse: High Confidence Software Reuse in Large Systems*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 166–177.

[111] H. Schirmeier and O. Spinczyk, "Challenges in software product line composition," in *System Sciences, 2009. HICSS '09. 42nd Hawaii International Conference on*, January 2009, pp. 1–7.

[112] M. O. Reiser, R. T. Kolagari, and M. Weber, "Compositional variability - concepts and patterns," in *Proceedings of the 42Nd Hawaii International Conference on System Sciences*. Washington, DC, USA: IEEE Computer Society, January 2009, pp. 1–10.

[113] A. Abele, Y. Papadopoulos, D. Servat, M. Törngren, and M. Weber, "The cvm framework - a prototype tool for compositional variability management," in *Fourth International Workshop on Variability Modelling of Software-Intensive Systems, Linz, Austria, January 27-29, 2010. Proceedings*, D. Benavides, D. S. Batory, and P. Grünbacher, Eds. Universität Duisburg-Essen, 2010, pp. 101–105.

[114] D. Dhungana, P. Grünbacher, R. Rabiser, and T. Neumayer, "Structuring the modeling space and supporting evolution in software product line engineering," *Journal of Systems and Software*, vol. 83, no. 7, pp. 1108–1122, 2010.

[115] J. Bröndum and L. Zhu, "Towards an architectural viewpoint for systems of software intensive systems," in *Proceedings of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge*. New York, NY, USA: ACM, 2010, pp. 60–63.

[116] J. Bröndum, "Software architecture for systems of software intensive systems (s3): the concepts and detection of inter-system relationships," in *2010 ACM/IEEE 32nd Interna-*

tional Conference on Software Engineering. New York, NY, USA: ACM, MAY 2010, pp. 355–356.

[117] A. Hubaux, A. Classen, and P. Heymans, "Formal modelling of feature configuration workflows," in *Proceedings of the 13th International Software Product Line Conference*. Pittsburgh, PA, USA: Carnegie Mellon University, 2009, pp. 221–230.

[118] C. Elsner, P. Ulbrich, D. Lohmann, and W. Schröder-Preikschat, *Consistent Product Line Configuration Across File Type and Product Line Boundaries*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 181–195.

[119] C. Elsner, D. Lohmann, and W. Schröder-Preikschat, "An infrastructure for composing build systems of software product lines," in *Proceedings of the 15th International Software Product Line Conference, Volume 2*. New York, NY, USA: ACM, 2011, pp. 18:1–18:8.

[120] S. Urli, "Model-driven composition of multiple software product lines," in *Proceedings of the 2Nd International Master Class on Model-Driven Engineering: Modeling Wizards*. New York, NY, USA: ACM, 2012, pp. 2:1–2:3.

[121] G. Holl, M. Vierhauser, W. Heider, P. Grünbacher, and R. Rabiser, "Product line bundles for tool support in multi product lines," in *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*. New York, NY, USA: ACM, 2011, pp. 21–27.

[122] V. T. Sarinho, A. L. A. Jr., and E. S. de Almeida, "Oofm - a feature modeling approach to implement mpls and dspls," in *2012 IEEE 13th International Conference on Information Reuse Integration (IRI)*. IEEE, August 2012, pp. 740–742.

[123] G. Holl, P. Grünbacher, C. Elsner, T. Klambauer, and M. Vierhauser, "Constraint checking in distributed product configuration of multi product lines," in *20th Asia-Pacific Software Engineering Conference (APSEC)*. Washington, DC, USA: IEEE Computer Society, December 2013, pp. 347–354.

[124] S. Urli, S. Mosser, M. Blay-Fornarino, and P. Collet, "How to exploit domain knowledge in multiple software product lines?" in *Product Line Approaches in Software Engineering (PLEASE), 2013 4th International Workshop on*. San Francisco, CA: IEEE, May 2013, pp. 13–16.

[125] R. Daniela, P. Herbert, G. Paul, P. Michael, E. Klaus, A. Florian, K. Mario, and G. Andreas, "Multi-purpose, multi-level feature modeling of large-scale industrial software system," *Software & Systems Modeling*, pp. 1–26, 2016.

[126] A. Kamoun, M. H. Kacem, and A. H. Kacem, "Multiple software product lines for service oriented architecture," in *2016 IEEE 25th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, June 2016, pp. 56–61.

[127] J. Bosch, "Software product lines: Organizational alternatives," in *Proceedings of the 23rd International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, May 2001, pp. 91–100.

[128] ——, "Toward compositional software product lines," *IEEE Softw.*, vol. 27, no. 3, pp. 29–34, May 2010.

[129] E. Y. Nakagawa and J. C. Maldonado, "Towards the open source reference architectures," in *Software Components, Architectures and Reuse (SBCARS), 2011 Fifth Brazilian Symposium on*, September 2011, pp. 61–70.

[130] I. Schaefer, L. Bettini, V. Bono, F. Damiani, and N. Tanzarella, "Delta-oriented programming of software product lines," in *Proceedings of the 14th International Conference on Software Product Lines: Going Beyond*. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 77–91.

[131] O. von Guericke-Universität Magdeburg, "Featureide," 2014. [Online]. Available: http://wwwiti.cs.uni-magdeburg.de/iti_db/research/featureide/

[132] U. de Hildesheim, "Easy-producer," 2015. [Online]. Available: http://sse.uni-hildesheim.de/en/fb4/institutes/ifi/software-systems-engineering-sse/research/projects/easy-producer/

[133] F. Damiani, I. Schaefer, and T. Winkelmann, "Delta-oriented multi software product lines," in *Proceedings of the 18th International Software Product Line Conference - Volume 1*. New York, NY, USA: ACM, 2014, pp. 232–236.

[134] C. Kästner, T. Thüm, G. Saake, J. Feigenspan, T. Leich, F. Wielgorz, and S. Apel, "Featureide: A tool framework for feature-oriented software development," in *Proceedings of the 31st International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 611–614.

[135] T. Thüm, C. Kästner, F. Benduhn, J. Meinicke, G. Saake, and T. Leich, "Featureide: An extensible framework for feature-oriented software development," *Science of Computer Programming*, vol. 79, pp. 70–85, 2014, experimental Software and Toolkits (EST 4): A special issue of the Workshop on Academic Software Development Tools and Techniques (WASDeTT-3 2010).

[136] J. A. Pereira, S. Krieter, J. Meinicke, R. Schröter, G. Saake, and T. Leich, *FeatureIDE: Scalable Product Configuration of Variable Systems*, ser. 9679. Springer International Publishing, 2016, ch. Software Reuse: Bridging with Social-Awareness.

[137] H. Eichelberger, S. El-Sharkawy, C. Kröher, and K. Schmid, "Easy-producer: Product line development for variant-rich ecosystems," in *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools - Volume 2*. New York, NY, USA: ACM, 2014, pp. 133–137.

[138] A. Classen, Q. Boucher, and P. Heymans, "A text-based approach to feature modelling: Syntax and semantics of tvl," *Science of Computer Programming*, vol. 76, no. 12, pp. 1130–1143, December 2011.

[139] D. Dhungana, P. Grünbacher, and R. Rabiser, "The dopler meta-tool for decision-oriented variability modeling: a multiple case study," *Automated Software Engineering*, vol. 18, no. 1, pp. 77–114, 2011.

[140] J. Bosch, *Maturity and Evolution in Software Product Lines: Approaches, Artefacts and Organization*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 257–271.

[141] G. Brunet, M. Chechik, S. Easterbrook, S. Nejati, N. Niu, and M. Sabetzadeh, "A manifesto for model merging," in *Proceedings of the 2006 international workshop on Global integrated model management - GaMMa '06*. New York, New York, USA: ACM Press, 2006, p. 5.

[142] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, and Y. Bontemps, "Generic semantics of feature diagrams," *Computer Networks*, vol. 51, no. 2, pp. 456–479, 2007.

[143] A. Gomez, L. Pibouleau, C. Azzaro-Pantel, S. Domenech, C. Latgé, and D. Haubensack, "Multiobjective genetic algorithm strategies for electricity production from generation {IV} nuclear technology," *Energy Conversion and Management*, vol. 51, no. 4, pp. 859 – 871, 2010.