DISEÑO DE UN SISTEMA GENÉRICO DE SOLVERS PARA EL MODELADO Y RAZONAMIENTO EN HERRAMIENTAS DE INGENIERÍA DE LÍNEAS DE PRODUCTOS

FRANCISCO JOSÉ PIEDRAHÍTA VÉLEZ

Director
PhD. RAUL MAZO PEÑA

REQUISITO PARA OPTAR AL TÍTULO DE MAGÍSTER EN INGENIERÍA

UNIVERSIDAD EAFIT
ESCUELA DE INGENIERÍA
MEDELLÍN - ANTIOQUIA - COLOMBIA
2023

AGRADECIMIENTOS

A mi querida familia, esta tesis de maestría es el resultado no solo de mis esfuerzos y dedicación, sino también de su amor incondicional, apoyo y sacrificios.

A mi asesor y mentor, Raúl Mazo, esta tesis no hubiera sido posible sin su guía, sabiduría y apoyo constante. Agradezco profundamente su paciencia y compromiso en mi formación académica.

RESUMEN

Las líneas de productos (PL, por sus siglas en inglés) constituyen entidades complejas, y para gestionar proyectos basados en esta metodología, se requieren herramientas que simplifiquen la enorme complejidad de configurar cientos de productos compuestos por miles de componentes. Para abordar estas dificultades, se han desarrollado diversas herramientas destinadas a la implementación de líneas de producto.

En la ingeniería de líneas de productos (PLE, por sus siglas en inglés), es común que ciertos casos de uso de estas herramientas requieran diferentes representaciones matemáticas que permitan evaluar los modelos que describen la línea de productos de diversas maneras. Uno de los principales desafíos en la PLE es habilitar un conjunto de herramientas lo suficientemente amplio como para modelar diferentes expresiones de modelos de PLE.

En este documento, exploramos una propuesta que permite añadir componentes para el razonamiento acerca de las propiedades de varios modelos de líneas de productos. Investigamos operaciones de razonamiento que no son tradicionales en la literatura de PLE. Además, examinamos cómo desarrollar un conjunto de procesos y artefactos de código que faciliten la creación de diferentes representaciones de los modelos y permitan razonar con diversas herramientas.

TABLA DE CONTENIDO

CAPÍTULO 1

INTRODUCCIÓN

1.1 - MOTIVACIÓN

La creciente complejidad y diversidad de los sistemas de software modernos ha creado la necesidad de nuevos enfoques para la ingeniería de software que puedan ayudar a las organizaciones a administrar y aprovechar sus activos de software de manera efectiva. Uno de estos enfoques es la ingeniería de líneas de productos. En la industria del software, esta nueva ingeniería se centra en el desarrollo de una familia de sistemas de software (para un segmento de mercado determinado) como un conjunto de variaciones a partir de un núcleo común. Este enfoque ofrece muchos beneficios potenciales, incluida la capacidad de reducir los costos de desarrollo, el tiempo de comercialización, mejorar la calidad del software y brindar mayor flexibilidad para adaptarse a las necesidades cambiantes de los usuarios**.**

A pesar de estos beneficios potenciales, la ingeniería de líneas de productos no está exenta de desafíos. Uno de los principales desafíos es la falta de herramientas y métodos efectivos para apoyar las diversas actividades de razonamiento involucradas en la ingeniería de líneas de productos, como el análisis de la comunalidad y la variabilidad, la derivación de productos, la gestión de las configuraciones y la verificación de los modelos de variabilidad y de las correspondientes configuraciones. Estas actividades requieren conocimientos y habilidades especializadas, y pueden ser tediosas y propensas a errores si se realizan manualmente**.**

Por lo tanto, hay una necesidad de nuevas herramientas y métodos para apoyar la ingeniería de líneas de productos. Estas herramientas y métodos deben ayudar a las organizaciones a superar los desafíos de la ingeniería de líneas de productos y a aprovechar al máximo el potencial de este enfoque. Esta tesina presenta un estudio del estado actual del arte en herramientas y métodos de ingeniería de líneas de productos, y propone un conjunto de nuevas técnicas para apoyar las actividades de razonamiento en el marco de la ingeniería de líneas de productos.

1.2 - PLANTEAMIENTO DEL PROBLEMA

Existen diversas herramientas para administrar las líneas de productos. Estas son utilizadas para gestionar miles de productos que a su vez tienen miles de componentes. Es normal que en el contexto de la ingeniería de líneas de productos sea necesario analizar estos productos y componentes desde diversas ópticas. Un ejemplo de esto lo podemos encontrar en una línea de productos de una compañía de automóviles, no es solo necesario analizar los productos de esa línea desde la economía o facilidad de ensamblaje, en adición debe analizarse la seguridad, el impacto climático y muchos otros factores. Estas facetas de una línea de productos requieren que ésta sea modelada con diversas representaciones, y a su vez cada una de estas representaciones requiere un solver que sea óptimo para ella.

El constante requerimiento de nuevos modelos y solvers hace que las herramientas existentes de líneas de software se vean limitadas, a razón de que en estas es extremadamente complejo adicionar estos nuevos modelos y solvers. Es por esta razón que es común observar que para modelar una misma línea de productos se requieran diferentes software. Esta multiplicidad de herramientas de modelado implica un costo

considerable en el desarrollo de los productos, ya que obliga a los ingenieros a ser versados en múltiples software y mantener en sincronía las diferentes representaciones de una misma línea de producto en cada uno de ellos.

A causa de las necesidades que estos desafíos generan, esta tesina se concentra en analizar la viabilidad de una herramienta genérica que permita la fácil integración de nuevos modelos y solvers a herramientas de líneas de productos, con la finalidad de reducir la complejidad de diseñar y analizar proyectos robustos de líneas de productos.

1.3 - PREGUNTAS DE INVESTIGACIÓN

- ¿En qué medida la herramienta permite la fácil integración de nuevos modelos y solvers de líneas de productos haciendo uso de componentes ya definidos?

- ¿Es posible generar una abstracción teórica que permita describir la mayoría de los modelos de las líneas de productos, junto con las operaciones de verificación que corresponden a cada modelo?

- ¿En qué medida la herramienta desarrollada puede ser integrada con otros sistemas de ingeniería de líneas de productos?

1.4 - HIPÓTESIS DE INVESTIGACIÓN

Es posible desarrollar una plataforma de software que permita describir modelos de líneas de productos e interactuar con solvers (herramientas de software que resuelven problemas matemáticos complejos) sin tener que estar familiarizado con sus detalles de implementación.

1.5 - CONTRIBUCIONES

En este documento se abordan elementos importantes en relación con la interacción que las herramientas de ingeniería de líneas de productos (PLE por sus siglas en Inglés) tienen con los solvers. Con la ambición de proponer una metodología novedosa para entender la relación de los modelos de líneas de productos y los solvers, simplificando los procesos de añadir nuevos modelos, solvers y operaciones de verificación con mínimo esfuerzo a herramientas de PLE.

En primer lugar se propone una **abstracción matemática que permite entender de manera formal los modelos de líneas de productos**. Esta abstracción matemática se basa en la representación de los modelos como un tipo particular de grafo. A partir de esta representación, se pueden definir operaciones formales para el razonamiento y análisis de los diferentes modelos.

Luego, se explora la **interacción entre los modelos de líneas de productos y los solvers**. Se discuten diferentes enfoques para integrar los solvers en herramientas de PLE, incluyendo el uso de interfaces estándar.

Finalmente, se propone una **metodología para simplificar el proceso de añadir nuevos modelos de líneas de producto, solvers y validaciones a herramientas de PLE**. Esta metodología se basa en la abstracción matemática propuesta anteriormente, y en la definición de un conjunto de interfaces estándar para la interacción entre los modelos de líneas de producto y los solvers. De esta manera, se busca facilitar la integración de nuevas funcionalidades en herramientas de PLE, y mejorar la productividad y eficiencia en el desarrollo de sistemas basados en líneas de productos.

CAPÍTULO 2

MARCO TEÓRICO

2.1 - INGENIERÍA DE LÍNEAS DE PRODUCTOS

Una línea de productos (PL por su sigla en Inglés) es un conjunto de productos relacionados entre sí que pertenecen al mismo segmento de mercado y son creados a partir de los activos de un dominio. **[1]** La construcción de una línea de productos se basa en la utilización de un conjunto de activos de ingeniería genéricos; es decir, diseñados de tal manera que puedan ser reutilizados para configurar y derivar cada uno de los productos que conforman la línea, de manera masiva.

Las líneas de productos ofrecen la posibilidad de personalización en masa gracias a la inclusión de mecanismos en los activos de dominio que permiten personalizar los requisitos, las arquitecturas, los componentes y los workflows de los productos dentro de la línea. **[2]** La ingeniería de líneas de productos (PLE) se refiere al conjunto de conocimientos, procesos, técnicas y herramientas que se emplean en la creación y mantenimiento de una línea de productos.

2.1.1 - PROCESO DE INGENIERÍA DE LÍNEAS DE PRODUCTOS

La ingeniería de líneas de productos define un proceso que describe la estructura conceptual básica necesaria para construir líneas de productos. A diferencia del proceso tradicional de ingeniería de software a la medida, el proceso de las líneas de productos

debe tener en consideración múltiples productos simultáneamente. **[1]** Para lidiar con esta complejidad adicional, la ingeniería de líneas de productos divide el procedimiento ingenieril en dos subprocesos: la ingeniería de dominio y la ingeniería de aplicaciones, ilustrados en la **Figura 1**.

La ingeniería de dominio está encargada de generar la plataforma común a partir de la cual se crearán los diferentes productos. Este subproceso tiene como objetivo identificar los activos comunes, al igual que los activos que pueden variar entre los diferentes productos que componen la línea. Luego de identificar dichos activos se crea un conjunto de componentes y de procesos de ingeniería que permiten la creación de una estructura basal para los productos que se pretende crear.

El segundo subproceso, llamado ingeniería de aplicaciones, se encarga de crear los diferentes productos a partir de la plataforma común de elementos ingenieriles. Este proceso se encarga de reutilizar de una forma metódica los componentes creados en los subprocesos pasados. La ingeniería de aplicaciones aprovecha la variabilidad inherente a las líneas de producto para asegurar la construcción adecuada de productos que sean capaces de satisfacer las necesidades específicas de sus usuarios.

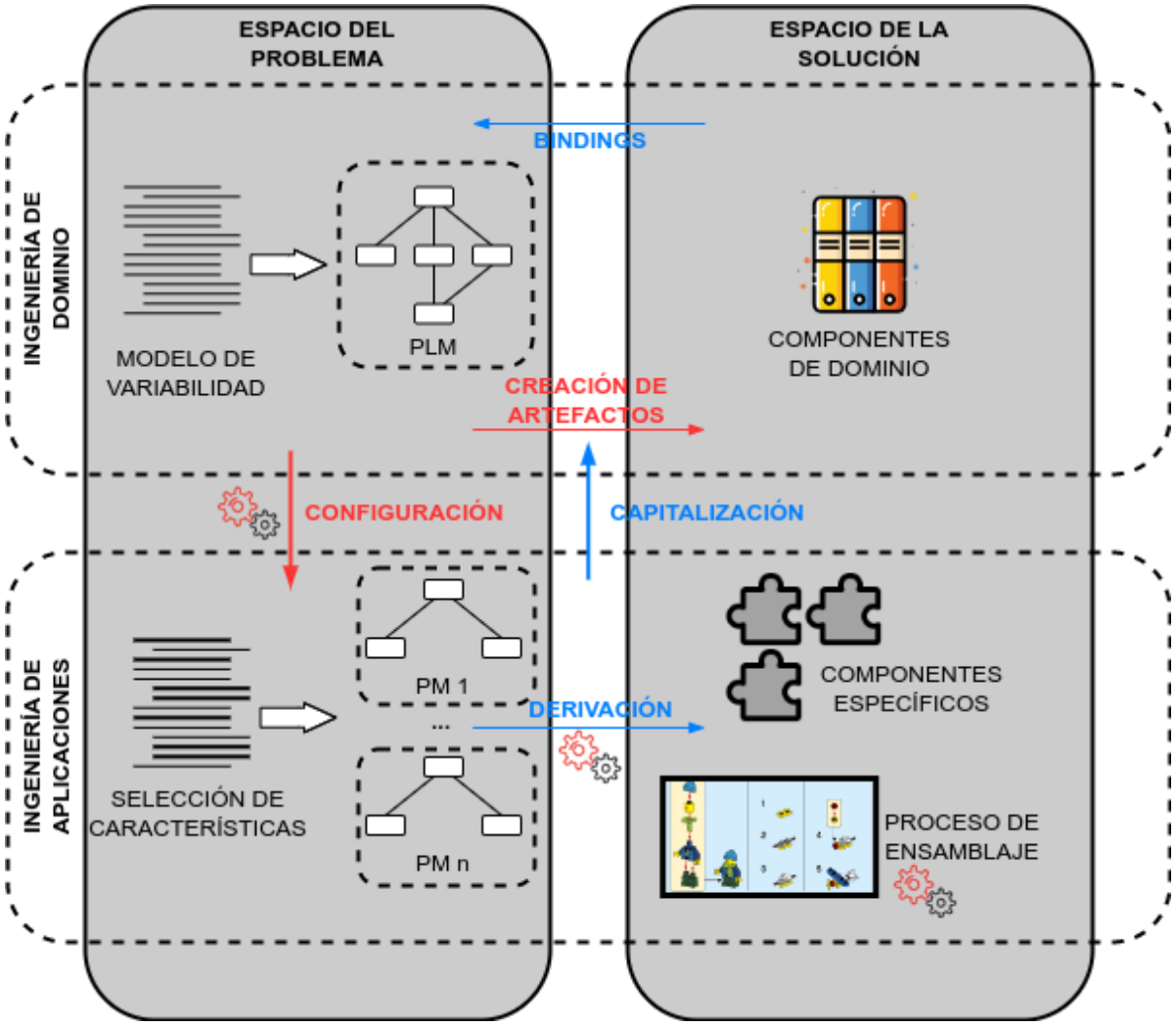

FIGURA 2.1: *Dos procesos de la ingeniería de líneas de productos desde las perspectivas del problema y de la solicitud.* **[1] [2]**

2.1.2 - VENTAJAS DE LA INGENIERÍA DE LÍNEAS DE PRODUCTOS

Las líneas de productos son un enfoque de ingeniería que permite a una organización desarrollar, mantener y evolucionar de manera eficiente una suite de productos. Una de las principales ventajas de utilizar un enfoque de línea de productos es la posibilidad de reutilizar componentes comunes de forma metódica y extensiva, lo que trae múltiples beneficios de costo, tiempos de comercialización y calidad. Las líneas de productos también permiten a las organizaciones adaptarse de manera más fácil a las necesidades cambiantes del mercado.

2.1.2.1 - REDUCCIÓN DE COSTOS

La ingeniería de líneas de productos puede reducir los costos de desarrollo **[1] [2]** de varias maneras. En primer lugar, al crear una plataforma común de elementos ingenieriles que puede ser utilizada para desarrollar múltiples productos, las organizaciones no tendrán que reinventar la rueda para cada nuevo producto, ahorrando una cantidad significativa de tiempo y esfuerzo. En segundo lugar, el uso de un conjunto común de elementos ingenieriles en toda la línea de productos permite una mejor utilización y coordinación de los recursos, lo que también puede generar ahorros de costos. Finalmente, la ingeniería de líneas de productos puede ayudar a las organizaciones a administrar y mantener sus productos de manera más eficiente a lo largo del tiempo, reduciendo los costos asociados con actualizaciones y mejoras. En general, la ingeniería de líneas de productos puede ayudar a las organizaciones a desarrollar y mantener sus softwares de manera más eficiente, lo que se traduce en menores costos.

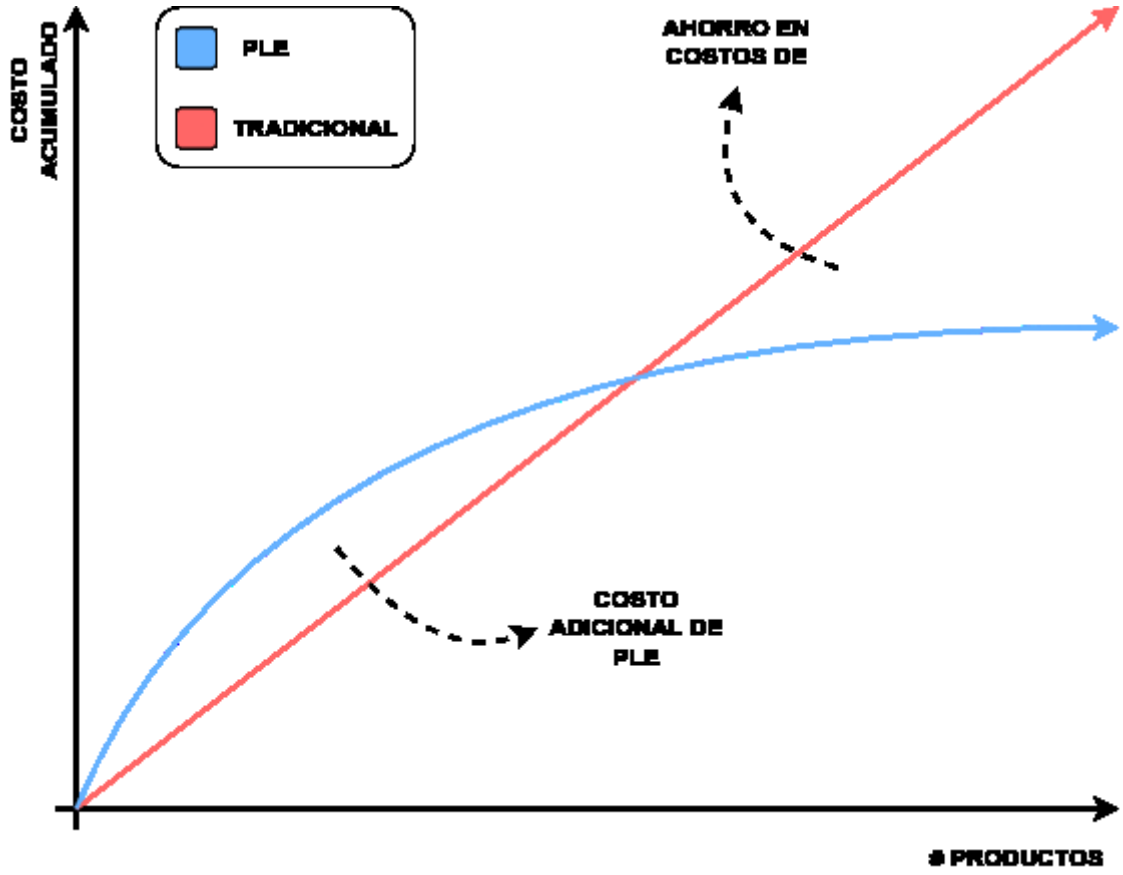

FIGURA 2.2: *Comparación de los costos de producción asociados a las estrategias de producción a la medida (rojo) y líneas de productos (azúl)* **[1] [2]**

La figura anterior representa el comportamiento de los costos de producción asociados a las líneas de productos en comparación con el método tradicional de desarrollo (producción a la medida). Podemos evidenciar cómo a medida que la cantidad de productos desarrollados aumenta, el costo acumulado de las aproximaciones tradicionales crece de forma lineal. Por otro lado, el precio de desarrollar los productos mediante líneas de producto tiene comportamiento asintótico horizontal, en el que el costo acumulado aumenta de forma menor cada vez que un producto es añadido. En los casos en que se desarrollan pocos productos, la aproximación tradicional genera costos acumulados menores, pero a medida que la cantidad de productos aumenta se hace más atractiva la alternativa de una línea de productos.

2.1.2.2 - REDUCCIÓN DE TIEMPO DE DESARROLLO

Uno de los principales beneficios de utilizar un enfoque de ingeniería de líneas de productos es que puede reducir el tiempo que lleva desarrollar nuevos productos. **[1] [2]** Esto se debe a que la ingeniería de líneas de productos permite la reutilización de componentes y funciones comunes. Al utilizar una plataforma de elementos ingenieriles existentes como punto de partida, los equipos de desarrollo pueden concentrarse en crear funcionalidades únicas para cada producto individual, en lugar de comenzar desde cero. Esto puede reducir significativamente la cantidad de tiempo y esfuerzo necesarios para llevar un nuevo producto al mercado. Además, el uso de una plataforma común puede ayudar a garantizar que los productos sean consistentes y de alta calidad, lo que también colabora a reducir el tiempo de comercialización, como se muestra en la **Figura 2.3**.

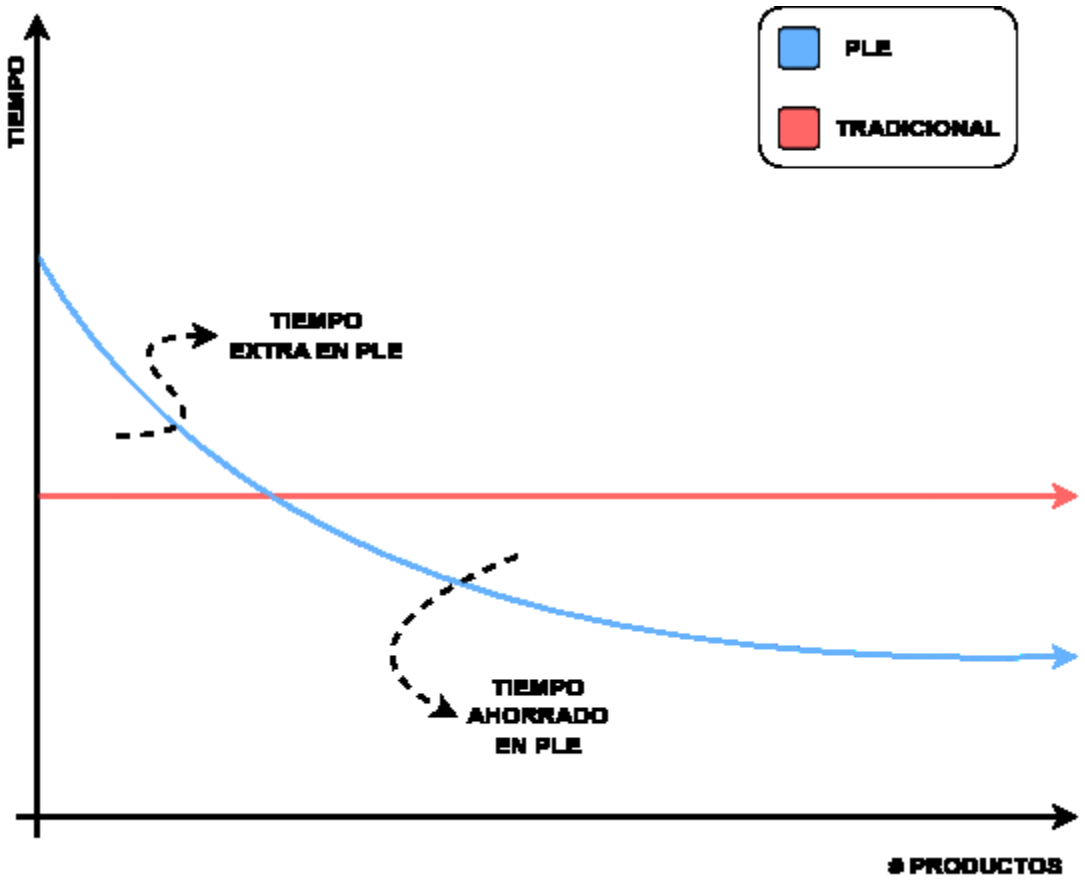

FIGURA 2.3: *Comportamiento del tiempo para sacar productos al mercado con las las estrategias de producción: a la medida (rojo) y lìneas de productos (azúl)* **[1] [2]**

De la figura anterior podemos inferir que el desarrollo tradicional de productos tiene un tiempo de creación constante. Lo anterior debido a que en esta metodología no se hace reutilización sistemática de los artefactos creados en productos anteriores. Por otro lado, podemos notar que el paradigma de líneas de productos tiene un comportamiento decreciente asintótico, en el que cada producto nuevo tiene un tiempo menor de desarrollo, hasta que el tiempo se estabiliza. Podemos notar también que cuando hay pocos productos las líneas de producto son más lentas que el desarrollo tradicional, esto se debe al trabajo adicional que implica la creación de una plataforma, pero una vez que existen estos artefactos comunes, los tiempos de las líneas de productos tienden a ser menores para cada producto nuevo.

2.1.2.3 - AUMENTO DE LA CALIDAD

Encontrar errores de forma temprana en el proceso de software es importante para mejorar la calidad de los productos, **[2] [3]** ya que entre más temprano se encuentre un defecto, menos es el costo de solucionarlo. La ingeniería de líneas de productos ayuda a encontrar defectos de manera más rápida. En primer lugar, la aproximación modular que esta metodología proporciona es de carácter planificado, esto hace que los errores de integración sean encontrados de forma más temprana. Por otro lado, los productos son generados a partir de modelos que están matemáticamente validados, permitiendo encontrar una mayor cantidad de defectos. **[2]** Adicionalmente, el hecho de que la plataforma común sea utilizada por múltiples productos hace que los errores que sean solucionados en los productos creados inicialmente se vuelvan garantía de que los productos posteriores no adolecerán del mismo error. Finalmente, la ingeniería de líneas de productos implementa técnicas para pruebas y validaciones automatizadas, particularmente para la verificación de variantes de productos, lo que puede ayudar a identificar y corregir errores de forma más eficiente. En general, los procesos de verificación, de validación y de testing en la ingeniería de líneas de productos pueden ayudar a mejorar la calidad del producto al facilitar la identificación temprana y la corrección de errores.

2.2 - PROBLEMA DE SATISFACCIÓN DE RESTRICCIONES

Un problema de satisfacción de restricciones (CSP) es un tipo de problema en el que el objetivo es encontrar una solución que pueda satisfacer un conjunto de restricciones. La solución debe garantizar que cada una de las restricciones sea válida cuando se reemplazan los valores de la solución.

Para definir de manera formal que es un CSP es conveniente analizar cada uno de los componentes matemáticos que forman este tipo de problema.

- **Dominio de una variable:** Es el conjunto de todos los posibles valores que pueden ser asignados a una variable. Si x es una variable, entonces podemos utilizar D_x para determinar el dominio de la misma. Un ejemplo de ésto puede ser una variable θ con dominio D_θ correspondiente a los enteros $\{1, 2, 3, \dots\}$.

- **Etiqueta:** Es una tupla que representa la asignación de un valor a una variable. Se utiliza (x, v) para indicar la etiqueta que asigna el valor v a la variable x. Es pertinente enunciar que (x, v) es válido si y sólo si $v \in D_x$.

- **Etiqueta compuesta:** Es una asignación simultánea de posibles valores a un conjunto de variables. Se utiliza $\{(x_1, v_1), (x_2, v_2), \dots, (x_n, v_n)\}$ para indicar la etiqueta compuesta de asignar $v_1, v_2, \dots, v_n$ a $x_1, x_2, \dots, x_n$ respectivamente.

- **Restricción:** Una restricción a un conjunto de variables es conceptualmente un conjunto de etiquetas compuestas para las variables del conjunto. Se puede expresar con C_S donde S es el conjunto de variables que se están restringiendo. Esto puede ser interpretado como una restricción al dominio de las variables, donde los valores de la etiqueta compuesta representan el nuevo dominio luego de la restricción.

- **Satisfacibilidad:** Si las variables de una etiqueta compuesta X son las mismas que las de los elementos de una restricción C, entonces X satisface C si y solo si X es un elemento de C, osea que $\{(x_1, v_1), (x_2, v_2), \dots, (x_n, v_n)\} \in C_{x_1, x_2, \dots, x_k}$.

Con estos elementos definidos, podemos decir que un problema de satisfacibilidad es una tripleta (Z, D, C), **[4]** donde:

- Z es un conjunto finito de variables $\{x_1, x_2, \ldots, x_n\}$.
- D es una función que transforma cada variable Z a un conjunto de objetos de tipo arbitrario, de la siguiente forma $D: Z -> \{D_1, D_2, \ldots, D_n\}$.
- C es un conjunto finito, posiblemente vacío, que contiene restricciones de un subconjunto de variables de Z. Esto hace que C sea un conjunto de conjuntos de etiquetas compuestas.

El objetivo final de un problema de satisfacibilidad de restricciones es asignar valores a cada variable de tal forma que todas las restricciones se cumplan. Un CSP es satisfacible si existe al menos un conjunto de soluciones que cumpla con el enunciado anterior.

El backtracking es un algoritmo particular que se utiliza para calcular si un conjunto de restricciones puede ser satisfacido. **[25]** El proceso de backtracking comienza con una solución parcial y, a continuación, busca posibles soluciones para las variables que aún no tienen un valor asignado. En cada paso, se prueba un valor para una variable y se comprueba si este valor satisface todas las restricciones. Si lo hace, se avanza a la siguiente variable; de lo contrario, se realiza una operación de retorno o backtrack para deshacer la última asignación y probar otro valor. Este proceso se repite hasta que se encuentra una solución completa que satisface todas las restricciones o se determina que no hay solución.

La propagación de chequeos es otro algoritmo utilizado para resolver CSPs. El algoritmo tiene como propósito evitar la propagación de conflictos futuros en el árbol de búsqueda. Cuando se asigna un valor a una variable en un problema de CSP, puede afectar las posibles soluciones de otras variables. **[25]** La propagación de chequeos o "forward checking" realiza un seguimiento de las restricciones que se aplican a cada variable y actualiza el dominio de cada variable en función de las asignaciones realizadas en las variables anteriores. Cuando se asigna un valor a una variable, la propagación hacia adelante realiza una comprobación de consistencia en las restricciones que involucran la variable asignada y las variables aún no asignadas, eliminando del dominio de estas últimas cualquier valor que entre en conflicto con la asignación de la variable actual.

Las otras soluciones conocidas para los problemas de satisfacción de restricciones se basan, en gran medida, en las técnicas de propagación de chequeos y backtracking, así como en sus variantes y mejoras. Estas técnicas permiten explorar el espacio de soluciones de manera eficiente y sistemática.

2.3 - SOLVERS

Un solver es una herramienta computacional diseñada para encontrar soluciones a problemas, generalmente con el propósito de optimizar alguna función objetivo. Los solvers son utilizados en una amplia gama de aplicaciones.

Existen muchos paradigmas de solvers, cada uno con sus propias características distintivas. Algunos están diseñados para encontrar soluciones exactas a problemas, mientras otros están diseñados para encontrar soluciones aproximadas que son lo suficientemente buenas para los propósitos particulares de su aplicación. Algunos están

diseñados para encontrar soluciones a problemas de forma rápida, mientras otros están diseñados para tardar más, pero encontrar una solución más precisa.

Uno de los principales problemas a la hora de elegir un solver para una aplicación en particular es equilibrar la disyuntiva entre eficiencia computacional y precisión de la solución. En muchos casos es posible encontrar soluciones a problemas de forma rápida, pero las soluciones pueden no ser tan precisas como las encontradas por métodos más intensivos computacionalmente.

Para encontrar un solver efectivo, es necesario analizar cuidadosamente las características del problema a resolver, así como las restricciones en la solución. Es posible también que diferentes facetas de un problema puedan beneficiarse de paradigmas de solvers diferentes, haciendo una necesidad poder tener más de una opción a la hora de elegir qué solver es adecuado para el problema particular.

Para nuestro caso los solvers que necesitaremos conceptualizar para fines del desarrollo de esta tesina son los que pueden representar problemas de satisfacibilidad. Existen múltiples paradigmas de solvers capaces de representar este tipo de problemas.

2.3.1 - SOLVERS DE SATISFACCIÓN BOOLEANA

Un solver de satisfacción booleana, o SAT por su abreviación en Inglés, es un algoritmo que tiene como entrada una fórmula Booleana y determina la existencia de una posible combinación de valores verdaderos o falsos para las variables de la fórmula que hagan que su evaluación sea verdadera. Los problemas SAT pueden ser entendidos como una forma especial de un CSP donde las variables son de dominio Booleano y las restricciones se construyen con operadores Booleanos. **[9]**

Este tipo de solvers también es utilizado para la solución de diversos problemas, como la verificación de correctitud de programas computacionales, las tareas de planificación, el razonamiento sobre sistemas complejos y la verificación de diseños de circuitos lógicos.

Es común que los problemas SAT sean escritos en forma normal conjuntiva (CNF). **[10]** CNF es un tipo especial de fórmula Booleana que está compuesta por una o más conjunciones de cláusulas y cada una de estas cláusulas puede ser una disyunción.

Un ejemplo de un problema que puede ser resuelto con un solver SAT es el de encontrar si una reunión puede ser hecha en cierto día de la semana teniendo en cuenta la disponibilidad de cada persona. **[11]** Supongamos las siguientes restricciones:

- Jhon puede reunirse uno de estos días: martes, miércoles o jueves.
- Catherine no puede reunirse los miércoles.
- Anne no puede reunirse los viernes.
- Peter no puede reunirse ni los martes ni los jueves.

La pregunta que queremos responder es: ¿Puede la reunión hacerse en algún día de la semana laboral? Para solucionar el problema lo podemos representar en CNF, quedando de la siguiente forma:

$$(Lun \vee Mie \vee Jue) \wedge (\neg Mie) \wedge (\neg Vie) \wedge \neg Mar \wedge \neg Jue$$

Este problema puede ser modelado con DIMACS CNF. **[10]** En esta representación cada día de la semana tiene un índice, siendo lunes el número uno y el viernes el cinco. Cada

espacio en una línea corresponde a una disyunción, mientras un cero acompañado de un cambio de línea representa una conjunción.

```
c encontrar_reunion.cnf
c
p cnf 5 5
1 2 4 0
-3 0
-5 0
-2 0
-4 0
```

Con nuestro archivo CNF creado, podemos utilizar CLASP[1] como solver para obtener el día de la semana en que la reunión se podría hacer. El siguiente comando demuestra cómo correr el solver haciendo uso del archivo CNF.

```
cat encontrar_reunion.cnf | clasp
```

Finalmente, CLASP retorna una solución en la que especifica si el problema es satisfacible, al igual que muestra las soluciones posibles. En este caso particular muestra que el día uno es posible hacer la reunión, mientras que en todos los otros no sería posible.

[1] CLASP es uno de los solvers SAT más populares. Viene incluido en casi todos los manejadores de paquetes de las diferentes distribuciones de linux. Por su reconocida popularidad y eficiencia se decidió utilizarlo para ejemplificar los solvers de tipo SAT. La versión utilizada fue la 3.9.9 que puede ser descargada desde el repositorio oficial del solver (https://github.com/potassco/clasp/releases/tag/v3.3.9).

```
c clasp version 3.3.9
c Reading from stdin
c Solving...
c Answer: 1
v 1 -2 -3 -4 -5 0
s SATISFIABLE
c
c Models         : 1
c Calls          : 1
c Time           : 0.001s (Solving: 0.00s 1st Model: 0.00s
Unsat: 0.00s)
c CPU Time       : 0.000s
```

Para entender el resultado que nos devuelve el solver tenemos que ver la línea que empieza con "V" que tiene a continuación el siguiente vector "1 -2 -3 -4 -5 0". En el vector cada número corresponde a un día de la semana laboral, siendo "1" lunes y "5" viernes. Los días con signo negativo son aquellos en los que no se puede hacer una reunión, mientras que los que no lo tienen son una opción válida para hacer la reunión. El cero al final representa el final del enunciado generado por el solver.

Los solvers CNF son importantes porque permiten resolver problemas booleanos de manera eficiente y concisa. La representación CNF de un problema booleano permite expresarlo en términos de una conjunción de cláusulas disyuntivas.

Los solver SAT son una excelente opción para resolver una amplia gama de problemas. A pesar de esto existen casos en los que estos solvers no se desempeñan de forma óptima. Particularmente en los problemas con un nivel elevado de redundancia o enunciados en los que se pretende encontrar el número de posibles soluciones, los solver SAT no se desempeñan de forma eficiente.

2.3.2 - SOLVERS DE DIAGRAMAS DE DECISIÓN BOOLEANA

Los diagramas de decisión Booleana (BDD por su sigla en inglés) son una técnica de representación y manipulación de fórmulas lógicas de manera compacta y eficiente, en forma de árbol binario. En particular, se utilizan en solvers para representar restricciones de problemas de satisfacibilidad y encontrar soluciones que satisfagan dichas restricciones mediante la manipulación de la información.

Una de las principales ventajas de los BDD es que permiten reducir el tamaño de la fórmula representada, lo que puede mejorar la eficiencia del solver al manipularla. Esto se logra gracias a una representación canónica de la fórmula que elimina redundancias y codifica la información de forma más compacta. **[17]**

La estructura de un BDD consiste en un grafo acíclico dirigido, en el que cada nodo puede ser un nodo de decisión etiquetado con una variable Booleana x_i con dos nodos hijo, o un nodo terminal etiquetado con FALSO o VERDADERO. Cada conexión de un nodo de decisión representa una asignación de FALSO o VERDADERO a una variable. La representación de la siguiente fórmula $\neg x_1 \wedge (x_2 \vee x_3)$ puede ser expresada a través del siguiente BDD.

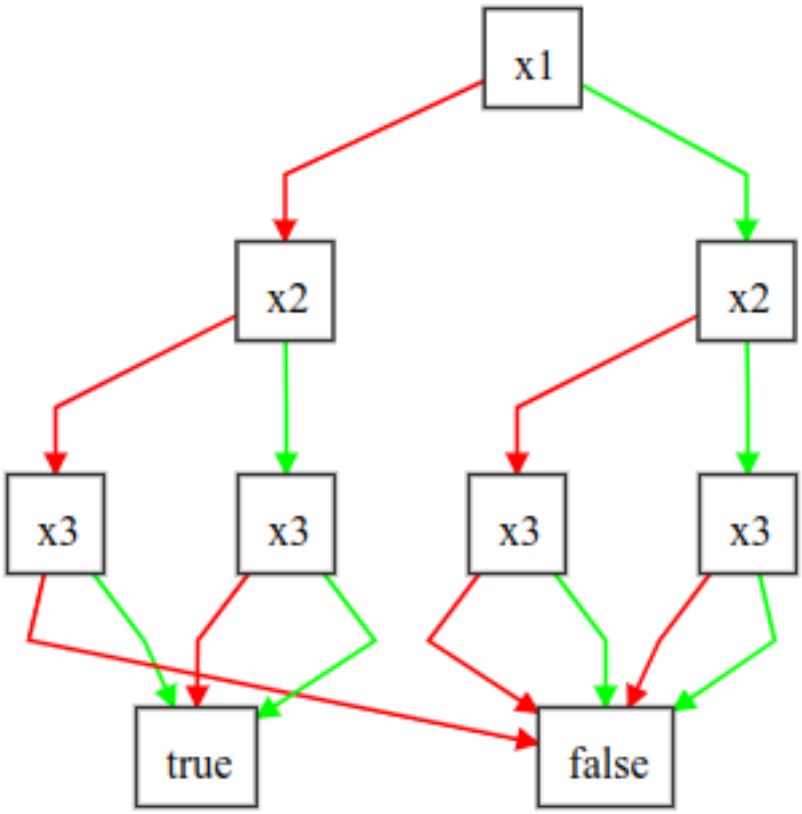

FIGURA 2.4: *Diagramas de decisión Booleana de la cláusula not x1 and (x2 or x3)*

Podemos utilizar el mismo escenario de coordinar una reunión dadas las agendas de quienes deben reunirse **(Sección 2.3.1)**. En este caso utilizaremos XCSP3[2] con PyCSP3[3] para modelar el problema.

```
from pycsp3 import *

lun = Var(dom={0, 1})
mar = Var(dom={0, 1})
```

[2] XCSP3 es un formato basado en XML diseñado para representar instancias de problemas combinatorios restringidos desde el punto de vista de la programación de restricciones (CP).

[3] PyCSP3 es una biblioteca de Python que nos permite escribir modelos de problemas combinatorios restringidos de manera declarativa. (https://pypi.org/project/pycsp3/2.1.2/)

```
mie = Var(dom={0, 1})
jue = Var(dom={0, 1})
vie = Var(dom={0, 1})

satisfy(
    ((lun == 1) | (mar == 1) | (jue == 1))
    & (mie == 0)
    & (vie == 0)
    & ((mar == 0) & (jue == 0))
)

if solve(sols=ALL, solver=SCOP) is SAT:
    print(f"NUM SOLUTIONS: {n_solutions()}")
```

El código define un problema de satisfacción booleana, en el que se buscan valores binarios para las variables lunes, martes, miércoles, jueves y viernes, que satisfagan ciertas restricciones. Estas restricciones son que al menos uno de los días lunes, martes o jueves debe tener un valor verdadero, mientras que miércoles y viernes deben tener un valor falso. Además, no se pueden asignar valores verdaderos a las variables martes y jueves al mismo tiempo. Luego se resuelve el SAT utilizando el solver SCOP[4], que es un solver BDD, y devuelve el número de soluciones encontradas para el problema.

Correr el programa es fácil, es solo cuestión de ejecutar el script de Python de forma normal.

```
python meeting_exaple.py
```

[4] SCOP es un solver que está implementado utilizando codificación BDD de los problemas SAT. Este solver es una reimplementación de Sugar con la adición de una capa BDD para deshacerse de las redundancias que pueden existir en el problema. (https://tsoh.org/sCOP/)

Al correr el programa se genera un archivo XML que corresponde a la compilación del código a XCSP3.

```
<instance format="XCSP3" type="CSP">
  <variables>
    <var id="lun"> 0 1 </var>
    <var id="mar"> 0 1 </var>
    <var id="mie"> 0 1 </var>
    <var id="jue"> 0 1 </var>
    <var id="vie"> 0 1 </var>
  </variables>
  <constraints>
    <intension>
      and(
        or(eq(lun,1),eq(mar,1),eq(jue,1)),
        eq(mie,0),
        eq(vie,0),
        eq(mar,0),
        eq(jue,0)
      )
    </intension>
  </constraints>
</instance>
```

Finalmente, correr el archivo de Python retorna que el problema tiene una única solución.

```
NUM SOLUTIONS: 1
```

Los BDD son una herramienta útil en una amplia variedad de escenarios, especialmente en problemas con un alto nivel de redundancia, ya que pueden eliminarlas. Además, los solvers que utilizan BDDs son particularmente eficientes en la determinación del número de posibles soluciones de un problema de satisfacción debido a la alta capacidad de

paralelismo permitida por la representación para encontrar caminos que conducen a "verdadero".

Tanto los solucionadores de tipo SAT como los de tipo BDD permiten modelar problemas con valores booleanos de manera eficiente. No obstante, en la práctica, existen problemas que requieren dominios más amplios, tales como números enteros o reales. Asimismo, dichos solucionadores no son capaces de abordar problemas en los cuales se necesita maximizar o minimizar una función objetivo. Por consiguiente, es necesario introducir otros tipos de solver que puedan ser utilizados en la resolución de escenarios que presenten estas características.

2.3.3 - SOLVERS DE PROGRAMACIÓN LINEAL ENTERA

La programación lineal entera (ILP por su sigla en inglés) es una clase de problema de optimización que busca encontrar el valor óptimo de un conjunto de variables de decisión sujetas a un conjunto de restricciones lineales, en las que una o todas las variables deben ser de tipo entero. La ILP es un caso especial de la programación lineal, que restringe el problema de tal manera que no pueden tomar valores fraccionarios. Esta técnica de programación tiene una amplia gama de aplicaciones en la investigación de operaciones y en las ciencias de la administración. **[14]**

En un problema de programación lineal, el objetivo es maximizar o minimizar una función objetivo lineal, dado un conjunto de variables de decisión y otro conjunto de restricciones lineales. Las variables se suelen representar mediante un vector $x = < x_1, x_2, \ldots, x_n >$, mientras que las restricciones se representan mediante una matriz A y un vector b, de modo que se cumpla $Ax \leq b$ o $Ax = b$. Se denomina región factible al conjunto de puntos que satisfacen todas las restricciones, que a su vez forman un poliedro convexo.

Para convertir un problema de programación lineal en un problema de programación lineal entera, es necesario añadir la condición de que las variables de decisión sean de tipo entero, de manera que $x_i \in \mathbb{Z}$ para todo i en los números enteros. Esta propiedad se puede garantizar mediante el uso de variables Booleanas o mediante variables enteras-mixtas, que pueden tomar un valor dentro de un rango específico.

El problema de la mochila es un problema clásico de optimización combinatoria que consiste en seleccionar un conjunto de elementos que tengan un valor total máximo, sujeto a una restricción de tamaño o peso total límite. Este problema se puede formular como una ILP de la siguiente manera: Dado un conjunto de elementos, cada uno con un peso w_i y un valor v_i, y una mochila con una capacidad máxima W, el objetivo es maximizar el valor total de los elementos seleccionados, denotado por V, sujeto a la restricción de que el peso total de los elementos seleccionados no exceda W. Las variables de decisión para este problema se pueden definir como $x_i \epsilon \{0, 1\}$, donde $x_i = 1$ si se selecciona el elemento i y $x_i = 0$ si no se selecciona el elemento i. **[15]**

MAXIMIZAR: $\sum \quad v_i * x_i$

SUJETO A: $\sum \quad w_i * x_i \leq W$

AND $x_i \epsilon \{0, 1\}$

Podemos resolver este problema de programación lineal entera (ILP) mediante el uso de un solver que utilice algoritmos como el método Simplex o el método Branch-and-bound para encontrar la solución óptima; la cual consiste en el conjunto de elementos con valor total máximo que cabe dentro de la capacidad de la mochila.

Por ejemplo, considere la siguiente instancia del problema de la mochila con 5 artículos y una capacidad de 10:

ITEM	PESO	VALOR
1	5	10
2	4	40
3	6	30
4	3	50
5	7	5

TABLA 2.1: *Problema de la mochila con 5 artículos*

Usando un solver ILP, podemos encontrar la solución óptima a este problema, que es $x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 1, x_5 = 0$, con un valor máximo de 90. Esta solución indica que el óptimo conjunto de elementos para seleccionar es $\{2, 4\}$, con un peso total de 7 y un valor total de 90.

Una herramienta popular para modelar este tipo de problema es MiniZinc. Con este lenguaje podemos expresar fácilmente un amplio conjunto de restricciones y funciones objetivo. Para modelar el problema definimos unas variables, luego un conjunto de restricciones, y finalmente una función objetivo. **[16]**

```
int: n; % number of objects
int: capacity;
array[1..n] of int: profit;
```

```
array[1..n] of int: size;

array[1..n] of var 0..1: x;

constraint sum(i in 1..n)(size[i] * x[i]) <= capacity;
solve maximize sum(i in 1..n)(profit[i] * x[i]);

output ["x = ", show(x), "\n"];
```

En este código, se declara una variable entera n que representa el número de objetos en el conjunto, y se definen dos arreglos: uno para los valores de cada objeto y otro para los tamaños de cada objeto. También se declara un arreglo de variables binarias x, donde cada variable indica si el objeto correspondiente está incluido en la mochila o no.

La restricción de capacidad se establece mediante una suma ponderada de los tamaños de los objetos seleccionados, y se asegura que esta suma no exceda la capacidad de la mochila. El objetivo del problema se define como la suma ponderada de los beneficios de los objetos seleccionados. Finalmente el código maximiza la suma de los valores para encontrar el número óptimo de objetos que puede cargar el morral que generen el máximo valor posible.

El problema modelado anteriormente es genérico, por lo que debemos definir una instancia particular del problema a resolver. Para esto asignamos valores concretos a cada una de las variables del problema.

```
n = 5;
capacity = 10;
profit = [10, 40, 30, 50, 5];
size = [5, 4, 6, 3, 7];
```

Cuando ya tenemos definidos los dos archivos anteriores podemos correr un solver utilizando el siguiente comando en la terminal.

```
minizinc pack.mzn  data.dzn
```

Finalmente el solver nos da un resultado, mostrando que para maximizar la ganancia, debemos elegir los artículos dos y cuatro para cargar en nuestro morral.

```
x = [0, 1, 0, 1, 0]
```

2.3.4 - SOLVERS DE PROGRAMACIÓN POR RESTRICCIONES

Un solver de programación por restricciones (CSP por su sigla en inglés) es un tipo de algoritmo utilizado para encontrar soluciones a problemas que involucran la satisfacibilidad de un conjunto de restricciones. **[5]**

Un solver CSP tiene como objetivo encontrar valores para un conjunto de variables que se ciñan a las restricciones definidas. Un ejemplo de un problema que puede ser resuelto con un solver CSP es el problema del mapa de cuatro colores, en este problema se define cada uno de los sectores del mapa como una variable que puede tomar uno de cuatro colores posibles, y la restricción es que no pueden existir dos sectores aledaños con los mismos colores.

FIGURA 2.5: *Mapa de Australia con sus diferentes estados.*

El siguiente código implementa un algoritmo de resolución del problema de coloreado de mapas aplicado al caso específico de Australia, utilizando XCSP3 para definir y resolver restricciones de satisfacción en un dominio de colores finito. Inicialmente se define una variable llamada "colors" que representa el conjunto de colores posibles. luego, se crean variables para cada uno de los estados de Australia Cada variable se asigna al dominio de posibles colores. Posteriormente, se establecen las restricciones del problema mediante la función "satisfy". Estas restricciones aseguran que dos estados adyacentes no compartan el mismo color, representadas como desigualdades entre las variables de los estados.

```
from pycsp3 import *

colors = data

western_australia = Var(range(colors))
northern_territory = Var(range(colors))
south_australia = Var(range(colors))
queensland = Var(range(colors))
new_south_wales = Var(range(colors))
victoria = Var(range(colors))
tasmania = Var(range(colors))

satisfy(
    western_australia != northern_territory,
    western_australia != south_australia,
    northern_territory != south_australia,
    northern_territory != queensland,
    south_australia != queensland,
    south_australia != new_south_wales,
    south_australia != victoria,
    queensland != new_south_wales,
    new_south_wales != victoria,
    victoria != tasmania
)

if solve() is SAT:
    print(solution().pretty_solution)
```

Con el problema modelado en PyCSP3 podemos proceder a compilarlo a XCSP3 que es un lenguaje genérico de restricciones. El proceso de compilación genera un archivo XML, que posteriormente es pasado a un solver, particularmente utilizaremos CHOCO en esta ocasión. Para esto podemos correr el siguiente comando en la terminal.

```
python map.py -data=4 -solver="CHOCO"
```

Finalmente, el solver retorna una respuesta en formato XML extendido. Esta respuesta nos dice si un óptimo pudo ser encontrado, acompañado de una lista de variables de salida junto con los valores que fueron asignados en la solución particular.

```
c Choco e747e1e
o 1 0.0
s OPTIMUM FOUND
v<instantiation id="sol1" type="solution">
v    <list>
v        western_australia
v        northern_territory
v        south_australia queensland
v        new_south_wales
v        victoria
v        Tasmania
v    </list>
v    <values> 0 1 2 0 1 0 1 </values>
v</instantiation>
d FOUND SOLUTIONS 1
```

También es posible ver el código compilado en una instancia particular de XCSP3. En él se ven las mismas restricciones que en el código python, pero ya acotadas a una instancia particular del problema en la que el número de colores es cuatro.

```
<instance format="XCSP3" type="CSP">
  <variables>
    <var id="western_australia"> 0..3 </var>
    <var id="northern_territory"> 0..3 </var>
    <var id="south_australia"> 0..3 </var>
```

```
    <var id="queensland"> 0..3 </var>
    <var id="new_south_wales"> 0..3 </var>
    <var id="victoria"> 0..3 </var>
    <var id="tasmania"> 0..3 </var>
  </variables>
  <constraints>
    <intension> ne(western_australia,northern_territory)
</intension>
    <intension> ne(western_australia,south_australia)
</intension>
    <intension> ne(northern_territory,south_australia)
</intension>
    <intension> ne(northern_territory,queensland) </intension>
    <intension> ne(south_australia,queensland) </intension>
    <intension> ne(south_australia,new_south_wales) </intension>
    <intension> ne(south_australia,victoria) </intension>
    <intension> ne(queensland,new_south_wales) </intension>
    <intension> ne(new_south_wales,victoria) </intension>
    <intension> ne(victoria,tasmania) </intension>
  </constraints>
</instance>
```

Los solvers de satisfacción de restricciones son comúnmente utilizados para solucionar una amplia gama de problemas, como la optimización de recursos, problemas de planificación, problemas de asignación de recursos, y muchos otros. Este tipo de solvers pueden ser muy eficientes para encontrar soluciones a problemas que involucran muchas variables y restricciones, y son especialmente adecuados para problemas que tienen un amplio espacio de búsqueda. **[6]**

Los solver analizados previamente en este estudio operan bajo un enfoque de "consulta-respuesta", en el cual se modela un problema y se espera obtener una solución. No obstante, en ciertos contextos, puede resultar ventajoso disponer de una interfaz que

permita realizar consultas incrementales o consecutivas. En tales situaciones, los paradigmas mencionados anteriormente podrían no ser suficientes para abordar eficazmente los desafíos planteados. Por consiguiente, es fundamental explorar enfoques alternativos que puedan satisfacer estas necesidades adicionales y ofrecer soluciones más versátiles y adaptables a problemáticas que requieran consultas de naturaleza incremental o consecutiva.

2.3.5 - SOLVERS DE PROGRAMACIÓN LÓGICA POR RESTRICCIONES

La programación lógica por restricciones (CLP por su sigla en inglés) es un paradigma de programación que combina los principios de la programación lógica con los de la programación por restricciones. **[12]** En este paradigma declarativo, el programador especifica un conjunto de restricciones que la solución a un problema debe satisfacer, en lugar de una secuencia de pasos a ejecutar. El solver de restricciones genera entonces una solución que satisface las restricciones dadas, de forma automática.

En CLP, los bloques básicos son las variables lógicas y las restricciones. Una variable lógica es un símbolo que puede tomar un valor determinado de un dominio. Mientras que una restricción es una expresión booleana que debe ser satisfecha por los valores asignados a las variables. En la programación lógica en general suelen existir dos pasos para escribir un programa, el primero es definir un conjunto de bloques lógicos llamados hechos, en segunda instancia se hacen consultas o preguntas que pueden ser respondidas con los hechos definidos anteriormente.

Podemos ver un ejemplo de cómo utilizar SWI Prolog, un lenguaje que permite modelar problemas CLP **[13]**. El problema SEND+MORE=MONEY es un ejemplo clásico de criptoaritmética que puede resolverse mediante la programación por restricciones. En

este problema cada letra representa un dígito único en una operación aritmética de suma. El objetivo es asignar un valor numérico específico a cada letra (S, E, N, D, M, O, R, Y) de tal manera que la suma SEND + MORE sea igual a MONEY, cumpliendo con la restricción de que cada letra represente un dígito diferente y que no haya dígitos repetidos en la solución.

```
:- use_module(library(clpfd)).

puzzle([S,E,N,D] + [M,O,R,E] = [M,O,N,E,Y]) :-
        Vars = [S,E,N,D,M,O,R,Y],
        Vars ins 0..9,
        all_different(Vars),
                  S*1000 + E*100 + N*10 + D +
                  M*1000 + O*100 + R*10 + E #=
        M*10000 + O*1000 + N*100 + E*10 + Y,
        M #\= 0, S #\= 0.
```

Para resolver el problema se utiliza la biblioteca clpfd, la cual provee herramientas y predicados especializados en programación por restricciones. Se define un predicado denominado "puzzle" que acepta como argumento la ecuación SEND+MORE=MONEY, representada mediante la suma de dos listas (SEND y MORE), cuyo resultado es una tercera lista (MONEY). A continuación, se construye una lista que contiene las variables correspondientes al problema.

Posteriormente, se impone la restricción de que todas las variables deben poseer valores distintos, empleando el predicado "all_different" provisto por la biblioteca clpfd. Seguidamente, se establecen restricciones adicionales a través de una ecuación aritmética. Es importante destacar que, en lugar de utilizar operadores aritméticos convencionales, se emplean operadores de restricción, como "#=", lo que permite a

Prolog trabajar con restricciones en lugar de valores específicos. De este modo, el código aborda eficientemente el problema de criptoaritmética mediante la implementación de técnicas de programación por restricciones en Prolog.

2.4 - ESTADO DEL ARTE

En esta sección se presenta un breve resumen de los diferentes enfoques que han sido utilizados para implementar herramientas de ingeniería de líneas de producto. El análisis abarca literatura desde el año 2005 hasta el año 2022, de donde se resaltan los referentes más importantes en el campo.

2.4.1 - FAMA

FAMA es una herramienta diseñada para el análisis automatizado de modelos de variabilidad, construida sobre la base del paradigma de líneas de producto. La herramienta, concebida inicialmente como un plugin de Eclipse, ha sido implementada en Java y se sustenta en tres paradigmas lógicos diferentes (CSP, SAT y BDD), cada uno de los cuales cuenta con su correspondiente solver (JaCoP, SAT4j y JavaBDD, respectivamente). **[18]**

Además de los solvers mencionados, FAMA ofrece un conjunto de herramientas útiles para el análisis de modelos de variabilidad, entre las que se incluyen un generador de modelos aleatorios y una herramienta de benchmarking. Asimismo, la herramienta permite la integración de nuevas funcionalidades a través de la creación de extensiones personalizadas, lo que aumenta su capacidad de adaptación a necesidades específicas.

2.4.2 - FeatureIDE

FeatureIDE es una herramienta de código abierto diseñada específicamente para facilitar el desarrollo y gestión de líneas de producto. Con el fin de proporcionar soporte para el análisis de dominio, esta herramienta pone a disposición editores gráficos para el modelado de variabilidad y permite la creación de modelos a gran escala gracias a su función de plegado. Además, FeatureIDE ofrece funciones para el análisis automatizado de modelos basados en SAT4J, permitiendo así la detección de anomalías en los modelos de características. **[19]**

FeatureIDE ofrece soporte para la integración con múltiples lenguajes de programación, entre ellos: Java, C, C++ y Haskell. Permitiendo así la generación de código con variabilidad en tiempo de ejecución por medio de parámetros y archivos de configuración. Además, esta herramienta proporciona varias funcionalidades adicionales como estadísticas de código, refactorización, documentación de código fuente con JavaDoc y especificación formal con JML.

2.4.3 - ProveLines

ProveLines es una herramienta para el análisis formal de modelos de líneas de producto. Los modelos en ProveLines están diseñados con una abstracción matemática denominada "Featured Transition Systems" (FTS por sus siglas en Inglés), que es una extensión de los sistemas de transición que representa el comportamiento de todos los productos en un solo modelo compacto. **[20]**

La herramienta también contiene un conjunto de algoritmos para verificar FTS contra propiedades expresadas en Lógica Temporal Lineal (LTL). Estos algoritmos aprovechan la información de variabilidad capturada en los FTS para verificar el comportamiento común

en varios productos sólo una vez, lo que reduce el tiempo de verificación. La plataforma utiliza SNIP, un verificador de SPL desarrollado desde cero por los autores de la herramienta.

2.4.4 - Pure::Variants

Pure::Variants es una herramienta de software que permite a los desarrolladores llevar el ciclo de vida de la variabilidad a través de múltiples productos de software. Pure::Variants cuenta con soporte para modelos con un gran número de componentes, un mecanismo de autorización a partir de roles, y la habilidad de crear informes a partir de los modelos de la línea de producto. La herramienta utiliza XML como formato único para el almacenamiento y transferencia de modelos. Pure::Variants fue desarrollado como un plugin de eclipse, en el lenguaje de programación C++. **[21]**

2.4.5 - SPLOT

SPLOT es una herramienta para el razonamiento y configuración de líneas de producto. El sistema está diseñado como una herramienta web, que permite a los usuarios interactuar con la plataforma en tiempo real y de forma eficiente. Las validaciones que SPLOT provee son realizadas a partir del paradigma lógico SAT y BDD.

Una particularidad de SPLOT es que disponibiliza un repositorio de modelos de líneas de producto. El repositorio contiene veinte modelos reales de líneas de producto, sacados de la literatura disponible. De la misma forma, el repositorio cuenta con más de diez mil modelos generados de forma automática por un algoritmo propuesto por los mismos autores de la herramienta.

2.4.6 - VariaMos

VariaMos es una herramienta que permite el modelado, integración, configuración y razonamiento de líneas de producto. Inicialmente la herramienta fue desarrollada como un plugin de eclipse, pero en versiones más recientes ha sido portada a entorno web, lo que permite a la herramienta ser de más fácil adopción. Los modelos en VariaMos son analizados por medio de transformaciones a un problema CSP, que luego se resuelve por medio del paradigma CLP con el solver SWI-Prolog. **[22]**

VariaMos difiere de las otras herramientas yo que permite que los modelos puedan ser validados de forma cruzada, asegurando que cada vista de la línea de productos sea consistente con todas las otras vistas. VariaMos también tiene una capa de transformación de modelos que permite que un solo modelo pueda ser traducido a diversas representaciones formales.

VariaMos busca ser una herramienta holística para el desarrollo y planificación de líneas de producto. Por esta razón VariaMos incluye herramientas adicionales que ninguna otra plataforma posee. Entre estas funcionalidades VariaMos cuenta con un módulo de estimación de costos para líneas de producto, al igual que tiene integración con el paradigma de programación por fragmentos que ayuda al mejor manejo de los puntos de personalización en las líneas de software. **[23]**

2.4.6 - PARADIGMAS DE SOLVER EN HERRAMIENTAS DE PLE

Es importante destacar que el número de paradigmas que utiliza cada herramienta difiere de caso en caso. El 21% de las publicaciones muestran no tener un solver, lo que representa a su vez que son herramientas que no podrán escalar a modelos de gran

tamaño. El 63% tienen al menos un paradigma, esto quiere decir que algunos modelos no podrán ser representados de la manera más eficiente posible. Tan solo el 16% de las herramientas de PLE utilizan más de un paradigma. Esto puede ser evidenciado en la **Figura 6**. **[24]**

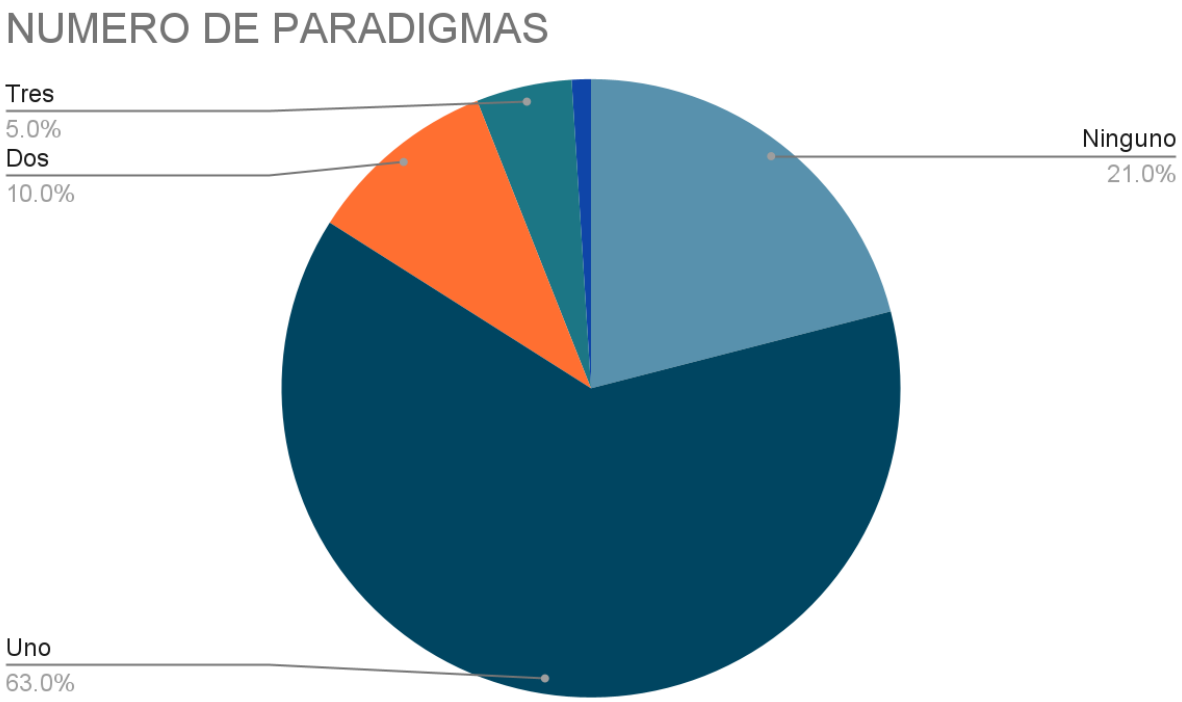

FIGURA 2.6: *Número de paradigmas lógicos utilizados en publicaciones de herramientas de PLE.* **[24]**

Por otro lado vemos también que la popularidad de cada paradigma es variable. SAT es el paradigma más popular en herramientas de PLE con 42 referencias en la literatura, aunque adolece a la hora de representar operaciones básicas lógicas como XOR. Sigue de forma cercana el paradigma de los CSP, con 40 estudios utilizándolo. Los paradigmas CLP, BDD y ILP suman 43 referencias. **[24]**

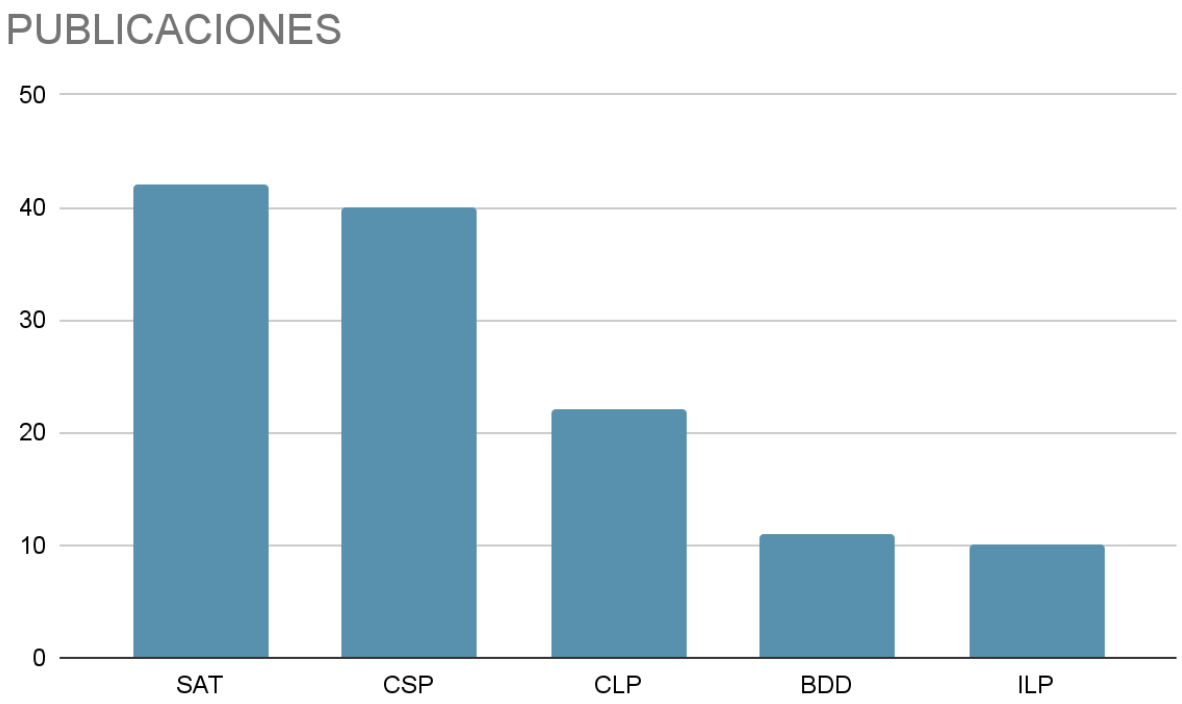

FIGURA 2.7: *Paradigmas lógicos utilizados en publicaciones de herramientas de PLE.* **[24]**

2.4.6 - SOLVERS EN HERRAMIENTAS DE PLE

Las herramientas de líneas de productos tienen una diversidad amplia en los solvers que utilizan. Es evidente que algunos solvers son predominantes en la literatura, como: SAT4J, CHOCO, GNU Prolog, JavaBDD. En general algo importante para anotar, es que la popularidad de los solvers decae rápidamente. Los cinco solvers más utilizados comprenden más del 60% de las apariciones en la literatura. Este comportamiento puede ser visto en la **Figura 8**. **[24]**

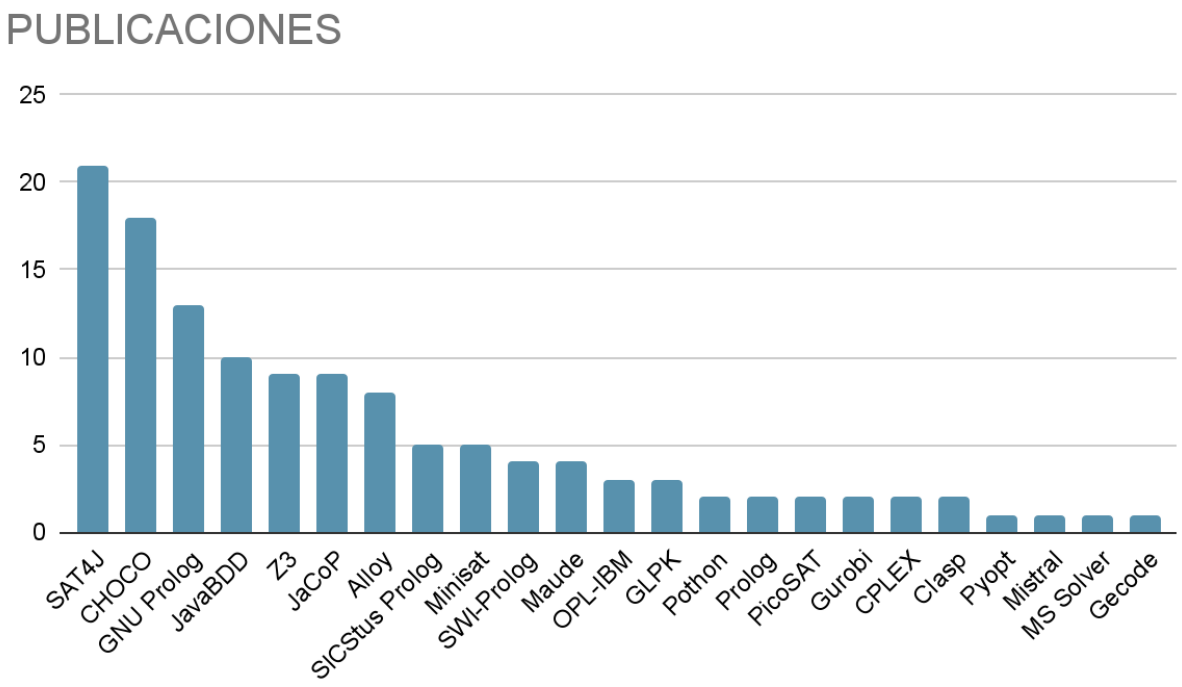

FIGURA 2.8: *Solvers utilizados en publicaciones de herramientas de PLE.* **[24]**

Un fenómeno singular que cabe destacar es que si sumamos las referencias a distintas versiones de Prolog, éste se convertiría en el solver más popular. Al revisar las particularidades de las investigaciones que citan el uso de estas variantes de Prolog, no encontramos razón alguna por la cual no pudieran ser implementadas todas ellas con una misma versión de Prolog.

CAPÍTULO 3

CASO DE PRUEBA

A continuación, se detalla un caso de prueba para la validación de la propuesta del backend genérico para líneas de productos que desarrolla el presente documento.

Se propone un caso de prueba en el que se analiza una línea de productos para la manufactura de teléfonos celulares. En primera instancia, se propone un modelo de características de la línea de productos de celulares. Y para concluir se presenta también un diagrama de transiciones para un producto de esta línea.

3.1 - MODELO DE CARACTERÍSTICAS

El modelo de características presentado es un ejemplo de los requisitos de una línea de productos de teléfonos móviles **(Figura 1)**, con el objetivo de capturar la variabilidad de características que podrían incluirse en estos productos. **[26]**

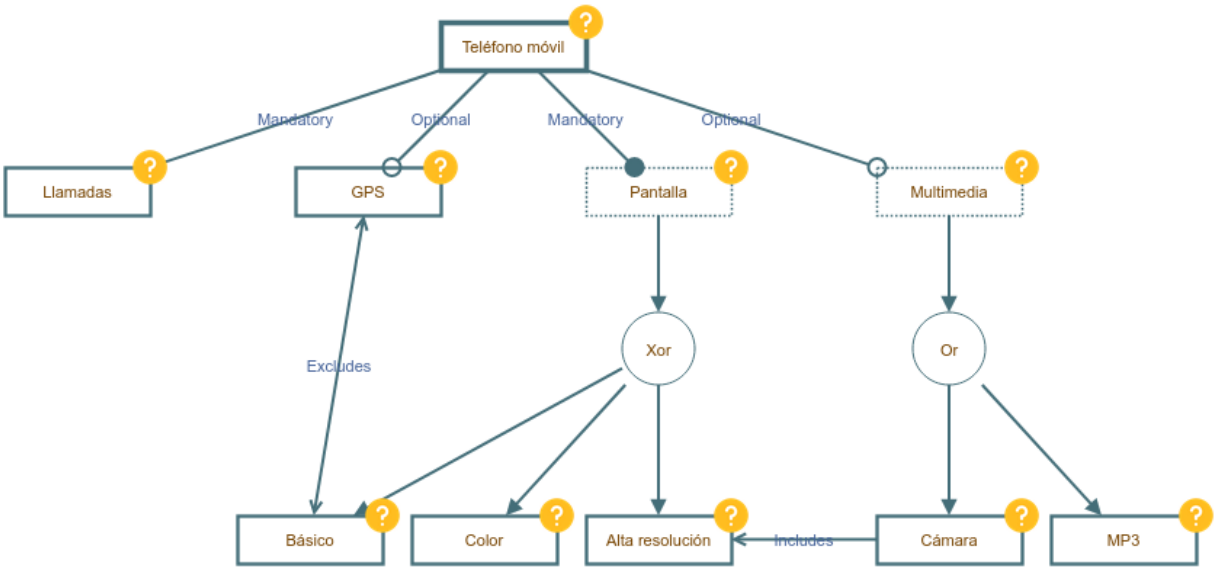

FIGURA 3.1: *Modelo de características de una línea de productos de celulares.* **[26]**

El modelo comprende relaciones obligatorias y opcionales entre las características y la entidad de Teléfono móvil. Específicamente, las características de Llamadas y Pantalla tienen una relación obligatoria con Teléfono móvil, lo que indica que todos los productos en la línea de teléfonos móviles deben incluir estas características. En contraste, las características de GPS y Multimedia tienen una relación opcional con Teléfono móvil, lo que significa que su inclusión, en los productos de la línea de teléfonos móviles, no es obligatoria.

Además, las características de Cámara y MP3 presentan una relación exclusiva con la característica Multimedia, lo que indica que siempre que se seleccione la característica Multimedia, al menos una de estas dos características también debe ser seleccionada. Por otro lado, las características de Básico, Color y Alta resolución presentan una relación alternativa con la característica Pantalla. Esto implica que en cualquier producto donde se encuentre presente la Pantalla, solo se puede seleccionar una de estas características.

El modelo incluye también relaciones transversales, que indican que la aparición de una característica depende de la presencia o ausencia de otras características. En particular, la característica de Cámara requiere la característica de Alta resolución, y la característica de GPS excluye la característica de pantalla Básica.

3.1 - DIAGRAMA DE TRANSICIONES

En una línea de productos, una máquina de estados se puede utilizar para modelar el comportamiento de un producto individual, lo que permite una mejor comprensión y análisis del comportamiento del producto en diferentes circunstancias.

Una máquina de estados es un modelo matemático que representa el comportamiento de un sistema como un conjunto de estados y las transiciones entre ellos. Al utilizar una máquina de estados, se puede analizar sistemáticamente el comportamiento de un producto bajo diversas condiciones de entrada y factores ambientales, lo que permite una mejor comprensión de su comportamiento.

La máquina de estados finitos tiene cinco estados: inactivo, menú, llamando, hablando y sonando. Cuando la máquina está en estado inactivo, puede pasar al estado menú si entra una llamada o si el usuario presiona la tecla de menú. También puede pasar al estado sonando si entra una llamada. Cuando la máquina está en el estado menú, puede pasar al estado inactivo si el usuario presiona la tecla de bloqueo o al estado llamando si se hace una llamada. Cuando está en el estado llamando, puede pasar al estado menú si se presiona la tecla de colgar o al estado hablando si se responde la llamada. Finalmente, cuando está en el estado hablando, puede pasar al estado inactivo si se cuelga la llamada. Si la máquina está en estado sonando, puede pasar al estado llamando si se contesta la

llamada o al estado inactivo si se cuelga la llamada. Se puede ver este comportamiento modelado en la **Figura 3.2**.

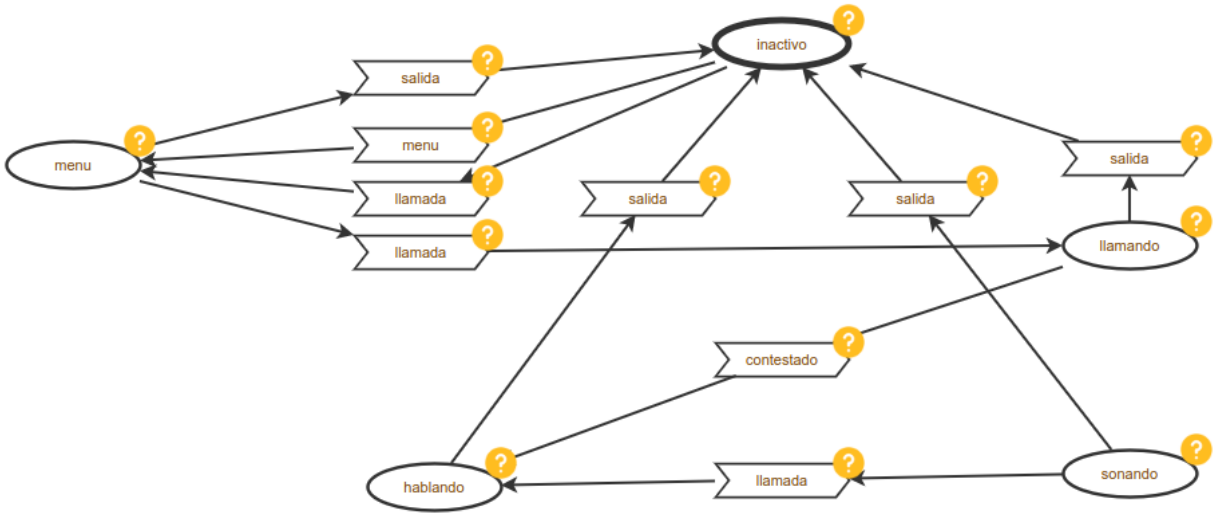

FIGURA 3.2: *Diagrama de transiciones de una línea de un celular.* **[27]**

CAPÍTULO 4

METODOS

En este capítulo se presenta la metodología usada para la implementación del sistema genérico de solvers, desde su etapa de diseño, hasta su puesta en funcionamiento y se ilustra cada etapa con el caso de prueba de conceptos presentado en el capítulo previo.

Para el diseño del dispositivo se hizo uso de la metodología de diseño ingenieril, con la intención de generar propuestas de diseño acordes con los requisitos técnicos del sistema en base a la información disponible en el estado del arte. Posteriormente con una propuesta de diseño, se procedió a su implementación. A continuación se explicará detalladamente cada una de estas etapas.

4.1 - PROCESO DE DISEÑO INGENIERIL

En ingeniería, el diseño es el proceso de conceptualización y especificación de diferentes componentes y funciones de un producto de ingeniería. El proceso de diseño en ingeniería está bien estudiado, y es fácil definir sus etapas, al igual que las entradas necesarias en cada una de las mismas. El siguiente diagrama **(Figura 1)** ilustra el flujo del proceso de diseño en productos de ingeniería. **[25]**

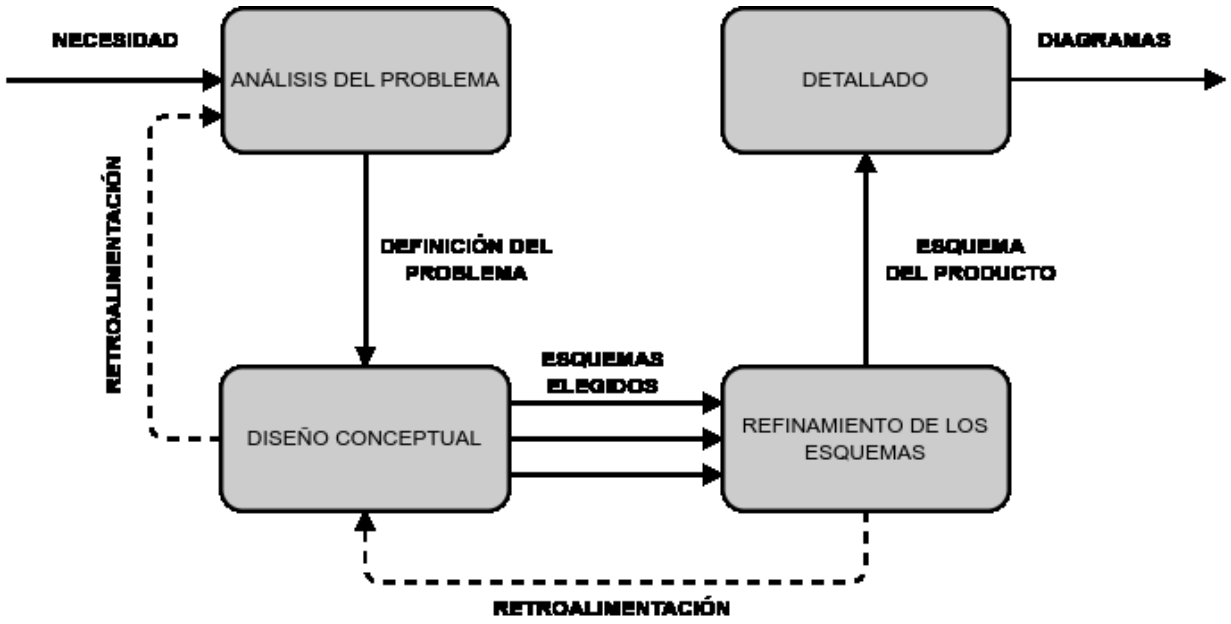

FIGURA 4.1: *Proceso de diseño ingenieril con sus fases y respectivas entradas y salidas.* **[28]**

El proceso ilustrado en la Figura 1 exhibe diversas ventajas, siendo la más sobresaliente la habilidad para adquirir contexto de manera ágil y continua. Es común encontrar fallas fundamentales en una idea de producto en etapas tardías de su desarrollo. Para minimizar este riesgo, el diseñador debe adquirir contexto tan rápido como le sea posible. Esto no es alcanzable por medio de la realización de cuidadosos y detallados estudios, si no por medio de la rápida iteración de ideas y su sucesiva mejora. En particular el proceso de diseño ingenieril está estructurado para permitir la veloz creación de ideas y prototipos que permitan evaluar las posibles alternativas de solución a los problemas.

4.2 - ANÁLISIS DEL PROBLEMA

La fase de análisis del problema es la primera etapa del proceso de diseño en ingeniería. Esta fase consiste en identificar el problema y las oportunidades existentes, para poder establecer los objetivos del proyecto.

Durante esta fase, el diseñador recopila información sobre el problema, para comprender mejor las necesidades y requisitos del proyecto. Esta información es generalmente recolectada por medio de investigaciones de mercado, entrevistas con las partes interesadas, análisis de datos de estudios existentes o por medio de la indagación de sistemas similares.

Con esta información, se elabora una definición del alcance del proyecto, en la cual se enumeran los objetivos que debería cumplir. Esto contribuye a la creación de una dirección clara para el proyecto, permitiendo al equipo evaluar diferentes conceptos e ideas en función de los objetivos previamente definidos.

La fase de análisis del problema es un paso esencial en el proceso de diseño, ya que garantiza que el proyecto se enfoque en resolver el problema adecuado, para así satisfacer las necesidades de las partes interesadas. También contribuye a la creación de un punto de partida para el trabajo de diseño, y simplifica el entendimiento entre el equipo de diseño y las partes interesadas.

La etapa de análisis del problema tiene como entrada un conjunto de necesidades, que en este caso serán recolectadas a través de opinión de expertos en el campo de las líneas de productos. Al finalizar esta etapa se obtiene una definición del problema, para nuestro caso particular esta definición del problema estará compuesta por los siguientes documentos: Especificación de requisitos, diagrama de estructura funcional y el modelo de calidad.

4.2.1 - ACTORES DE LA PLATAFORMA

En este capítulo, se describen los tres actores principales de la plataforma: el usuario, el investigador y el desarrollador. Cada actor tiene sus propias necesidades y requisitos específicos en términos de funcionalidades y herramientas de la plataforma para realizar sus tareas de manera efectiva.

El primer actor de interés en la plataforma es el usuario, un agente que busca modelar una línea de productos en un contexto industrial con el objetivo de fabricar productos a partir de este modelo. Este actor utiliza la plataforma para crear y gestionar modelos, simular y analizar los resultados, y optimizar el modelo para obtener los mejores resultados posibles.

El segundo actor relevante en la plataforma es el investigador, un experto en la implementación y modelado de líneas de productos que, aunque carece del conocimiento específico en programación, busca evaluar el rendimiento de diferentes solvers para un mismo modelo y encontrar el solver que más convenga para el modelo especificado. Las historias de usuario que ejemplifican los requisitos funcionales que este actor espera del sistema incluyen la capacidad de evaluar y comparar el rendimiento de los solvers, y la posibilidad de personalizar los solvers para satisfacer sus necesidades específicas.

El tercer y último actor relevante en la plataforma es el desarrollador, un usuario con conocimientos técnicos especializados en la plataforma y en la implementación particular de cada solver. Este actor tiene como objetivo principal la ampliación de las opciones de solvers disponibles en la plataforma para ofrecer mayores posibilidades y alternativas de resolución de problemas a los demás actores implicados.

4.2.2 - REQUISITOS FUNCIONALES

Los requisitos funcionales son capacidades que un sistema debe tener para cumplir su propósito. Este tipo de objetivos suele definirse en términos de las metas del sistema y pueden incluir tareas específicas que el sistema debe realizar. Los requisitos funcionales son un elemento clave del proceso de diseño y desarrollo de un sistema de software ya que ayudan a garantizar que el producto final cumpla las necesidades de los usuarios. En el caso particular de este trabajo utilizaremos historias de usuario para definir los requisitos funcionales del sistema.

ID: RF-1	**ACTOR:** Usuario
DESCRIPCIÓN: Como usuario, quiero una herramienta que me permita analizar diferentes facetas de una línea de productos, para poder modelar formalmente cada factor de interés de la línea.	

TABLA 4.1: *Requisito funcional RF-1*

ID: RF-2	**ACTOR:** Usuario
DESCRIPCIÓN: Como usuario, quiero poder modelar diferentes representaciones de una línea de productos en una sola herramienta, para poder tener una vista global de la misma.	

TABLA 4.2: *Requisito funcional RF-2*

ID: RF-3	**ACTOR:** Usuario

DESCRIPCIÓN:
Como usuario, quiero tener acceso a los solvers óptimos para cada representación de la línea de productos, para poder obtener resultados precisos en tiempos razonables.

TABLA 4.3: *Requisito funcional RF-3*

ID: RF-4	**ACTOR:** Usuario
DESCRIPCIÓN:	
Como usuario, quiero una herramienta que me permita modelar líneas de productos con miles de componentes, para poder gestionar proyectos industriales de alto impacto.	

TABLA 4.4: *Requisito funcional RF-4*

ID: RF-5	**ACTOR:** Usuario
DESCRIPCIÓN:	
Como usuario, quiero poder visualizar los diferentes solvers y modelos disponibles para poder elegir los que mejor se adaptan a mis necesidades.	

TABLA 4.5: *Requisito funcional RF-5*

ID: RF-6	**ACTOR:** Usuario
DESCRIPCIÓN: Como usuario, quiero poder diseñar mis propios modelos y el modelo matemático que los representa para poder utilizarlos en el modelado de mis líneas de productos.	

TABLA 4.6: *Requisito funcional RF-6*

ID: RF-7	**ACTOR:** Investigador
DESCRIPCIÓN: Como investigador, quiero poder evaluar el rendimiento de diferentes solvers para el mismo modelo para poder encontrar el solver que más convenga para el modelo especificado.	

TABLA 4.7: *Requisito funcional RF-7*

ID: RF-8	**ACTOR:** Investigador
DESCRIPCIÓN: Como investigador, quiero poder generar diferentes representaciones matemáticas del modelo para poder hacer investigación de diferentes metodologías de representación matemática de los modelos.	

TABLA 4.8: *Requisito funcional RF-8*

ID: RF-9	**ACTOR:** Investigador
DESCRIPCIÓN: Como investigador, quiero tener la posibilidad de crear nuevos modelos, acompañados de su traducción matemática, y sus operaciones de razonamiento para evaluar la utilidad de nuevos modelos en diferentes facetas del modelado de líneas de productos.	

TABLA 4.9: *Requisito funcional RF-9*

ID: RF-10	**ACTOR:** Desarrollador
DESCRIPCIÓN: Como desarrollador quiero que la herramienta genérica tenga una arquitectura modular, para poder agregar nuevos modelos y solvers de manera sencilla.	

TABLA 4.10: *Requisito funcional RF-10*

ID: RF-11	**ACTOR:** Desarrollador
DESCRIPCIÓN: Como desarrollador, quiero que la herramienta tenga interfaces bien definidas para que al momento de agregar un solver, tenga seguridad que todas las funciones necesarias fueron definidas.	

TABLA 4.11: *Requisito funcional RF-11*

ID: RF-12	**ACTOR:** Desarrollador

DESCRIPCIÓN:
Como desarrollador, quiero que la herramienta genérica tenga una documentación clara, para poder agregar nuevos solvers con facilidad y agilidad.

TABLA 4.12: *Requisito funcional RF-12*

4.2.3 - REQUISITOS NO FUNCIONALES

Los requisitos no funcionales son un tipo de requisito que especifica cómo el sistema debe desempeñarse en vez de describir las diferentes tareas que el sistema debe ejecutar. Los requisitos no funcionales están profundamente relacionados con los modelos de calidad, ya que éstos describen propiedades de cómo los sistemas se desempeñan. Los modelos de calidad son utilizados para entender el grado en que un sistema cumple con sus requisitos no funcionales.

De forma clara los requisitos no funcionales son fundamentales para definir la arquitectura de un sistema de software. Un claro ejemplo de esto son los sistemas que tienen como requisito no funcional completar sus operaciones por debajo de un tiempo límite de respuesta. En este caso se tomarán decisiones que beneficien los tiempos de respuesta, como elegir un lenguaje de alto rendimiento como es Rust, o utilizar estructuras de datos altamente optimizadas.

Es importante definir claramente los requisitos no funcionales del sistema, ya que éstos determinan gran cantidad de las decisiones importantes de diseño de software, que a su vez impactan las herramientas que se utilizarán y los algoritmos que se deben implementar.

Es común definir los requisitos funcionales de un sistema utilizando un modelo de calidad, ya que éstos definen un conjunto de características y métricas que permiten estudiar los sistemas. Como modelo de calidad para esta tesina, hemos elegido ISO/IEC 25000, también conocido como SQuaRE. Este modelo está compuesto por una serie de estándares internacionales para la calidad del producto de software. Dichos estándares proporcionan un marco común para evaluar la calidad de los productos de software **[29]** y son utilizados por organizaciones de todo el mundo para este efecto y para identificar áreas de mejora.

A fin de ilustrar las características empleadas para evaluar el sistema, se han elaborado diversas tablas. Cada tabla representa una característica de calidad, es decir, un atributo inherente al producto de software que incide en su calidad global. Además, se incluyen en cada tabla las métricas correspondientes a cada característica, las cuales constituyen indicadores cuantitativos que facilitan la medición y evaluación de dichas características de calidad.

CARACTERÍSTICA DE CALIDAD	
ID: NFR-1	**NOMBRE:** Adecuación Funcional
DESCRIPCIÓN: El concepto de adecuación funcional, en relación a un software determinado, se refiere al grado en que dicho programa logra satisfacer tanto las expectativas explícitas como implícitas de las diversas partes interesadas en su uso, en distintos contextos y situaciones específicas.	
METRICAS	

ID: M-1	**NOMBRE:** Cobertura Funcional
DECRIPCIÓN: Consiste en una comparación sistemática entre los requisitos funcionales establecidos por los diferentes actores involucrados y las funcionalidades efectivamente provistas por el software en cuestión. Este enfoque analítico tiene por finalidad medir y evaluar el grado de correspondencia existente entre las necesidades y expectativas de los usuarios y las capacidades del programa.	**FORMULA:** $$1 - \frac{No.\,funciones\ faltantes}{No.\,funciones\ especificadas}$$
ID: M-2	**NOMBRE:** Correctitud Funcional
DECRIPCIÓN: La métrica en cuestión tiene por finalidad evaluar la proporción de funciones dentro del sistema en cuestión, que efectivamente retornan el resultado correcto en relación al total de funciones que conforman dicho sistema.	**FORMULA:** $$1 - \frac{No.\,funciones\ incorrectas}{No.\,total\ de\ funciones}$$
ID: M-3	**NOMBRE:** Cobertura de Solvers Ajustada Por Popularidad

DECRIPCIÓN:	**FORMULA:**
La métrica en cuestión se compone de la comparación cuantitativa entre el número de solvers que se encuentran disponibles en la plataforma y los solvers existentes en la literatura especializada. Esta evaluación se lleva a cabo mediante un ponderado basado en la frecuencia con que cada solver ha sido mencionado en dicha literatura. Esta métrica resulta relevante en el ámbito académico y científico para evaluar la calidad y eficiencia de las plataformas de ingeniería de líneas de productos.	$\frac{\sum_{i=1}^{n} Solver_i \times Referencia_i}{\sum_{i=1}^{n} Referencia_i}$

TABLA 4.13: *Requisito no funcional RNF-1*

CARACTERÍSTICA DE CALIDAD	
ID: NFR-2	**NOMBRE:** Eficiencia de Desempeño
DESCRIPCIÓN: La eficiencia de desempeño se define como una característica fundamental de calidad de software, que alude a la cantidad de trabajo de valor que un sistema de software puede realizar, dadas ciertas condiciones operativas y un conjunto de recursos determinados.	
METRICAS	
ID: M-4	**NOMBRE:** Tiempo Medio de Respuesta
DECRIPCIÓN: El tiempo medio de respuesta es una métrica que tiene por objetivo cuantificar el tiempo que un sistema informático o tecnológico requiere para proporcionar una respuesta al usuario, en condiciones normales de operación.	**FORMULA:** $\sum \frac{Tiempo\ de\ respuesta}{No.\ de\ respuestas}$

TABLA 4.14: *Requisito no funcional RNF-2*

<table>
<tr><th colspan="2">CARACTERÍSTICA DE CALIDAD</th></tr>
<tr><td>ID: NFR-3</td><td>NOMBRE: Fiabilidad</td></tr>
<tr><td colspan="2">DESCRIPCIÓN:

La fiabilidad es una característica de calidad del software que se define por su capacidad para desempeñar una tarea específica en un lapso de tiempo determinado, bajo ciertas condiciones operativas, sin sufrir degradación o fallas significativas.</td></tr>
<tr><th colspan="2">METRICAS</th></tr>
<tr><td>ID: M-5</td><td>NOMBRE: Cobertura de Pruebas</td></tr>
<tr><td>DECRIPCIÓN:

Esta métrica tiene como propósito medir la proporción entre el número de funciones probadas y el total de funciones implementadas en un sistema de software determinado.</td><td>FORMULA:

$$\frac{No.\,funciones\ probadas}{No.\,total\ de\ funciones}$$</td></tr>
<tr><td>ID: M-6</td><td>NOMBRE: Adecuación de Complejidad Ciclomática</td></tr>
<tr><td>DECRIPCIÓN:

La complejidad ciclomática es una métrica de software que mide la cantidad de rutas</td><td>FORMULA:</td></tr>
</table>

de ejecución posibles en un programa, indicando su complejidad y facilitando la identificación de riesgos de errores. La adecuación de complejidad ciclomática se define como la estimación del número de funciones que cumplen con un valor predeterminado de complejidad ciclomática, en relación al total de funciones existentes en un sistema de software determinado.	$\frac{No.\,funciones\ con\ CC\ <\ CC\ predefinid}{No.\,total\ de\ funciones}$
ID: M-7	**NOMBRE:** Conformidad a Reglas de Código
DECRIPCIÓN: La métrica en cuestión se enfoca en la evaluación del nivel de conformidad de una base de código con los estándares de codificación definidos para un proyecto determinado.	**FORMULA:** $\frac{No.\,funciones\ que\ cumplen}{No.\,total\ de\ funciones}$
ID: M-8	**NOMBRE:** Completitud de Funciones probadas
DECRIPCIÓN: La completitud de funciones probadas se	**FORMULA:**

define como la medición del grado de exhaustividad con el que un sistema de software ha sido probado.	$\frac{No.\,funciones\,probadas}{No.\,total\,de\,funciones}$

TABLA 4.15: *Requisito no funcional RNF-3*

Con estas características de calidad, acompañadas de sus respectivas métricas, nos permiten evaluar la propuesta de esta tesina. Los requisitos previamente definidos permiten determinar la capacidad del sistema para cumplir con las expectativas de los diferentes actores del sistema.

4.2.5 - ESTRUCTURA FUNCIONAL

Mediante un análisis detallado de los flujos del sistema y una estricta adhesión a los requisitos de diseño, se plantea una alternativa de estructura funcional **(Figura 2)**, que satisface los requisitos de diseño previamente establecidos.

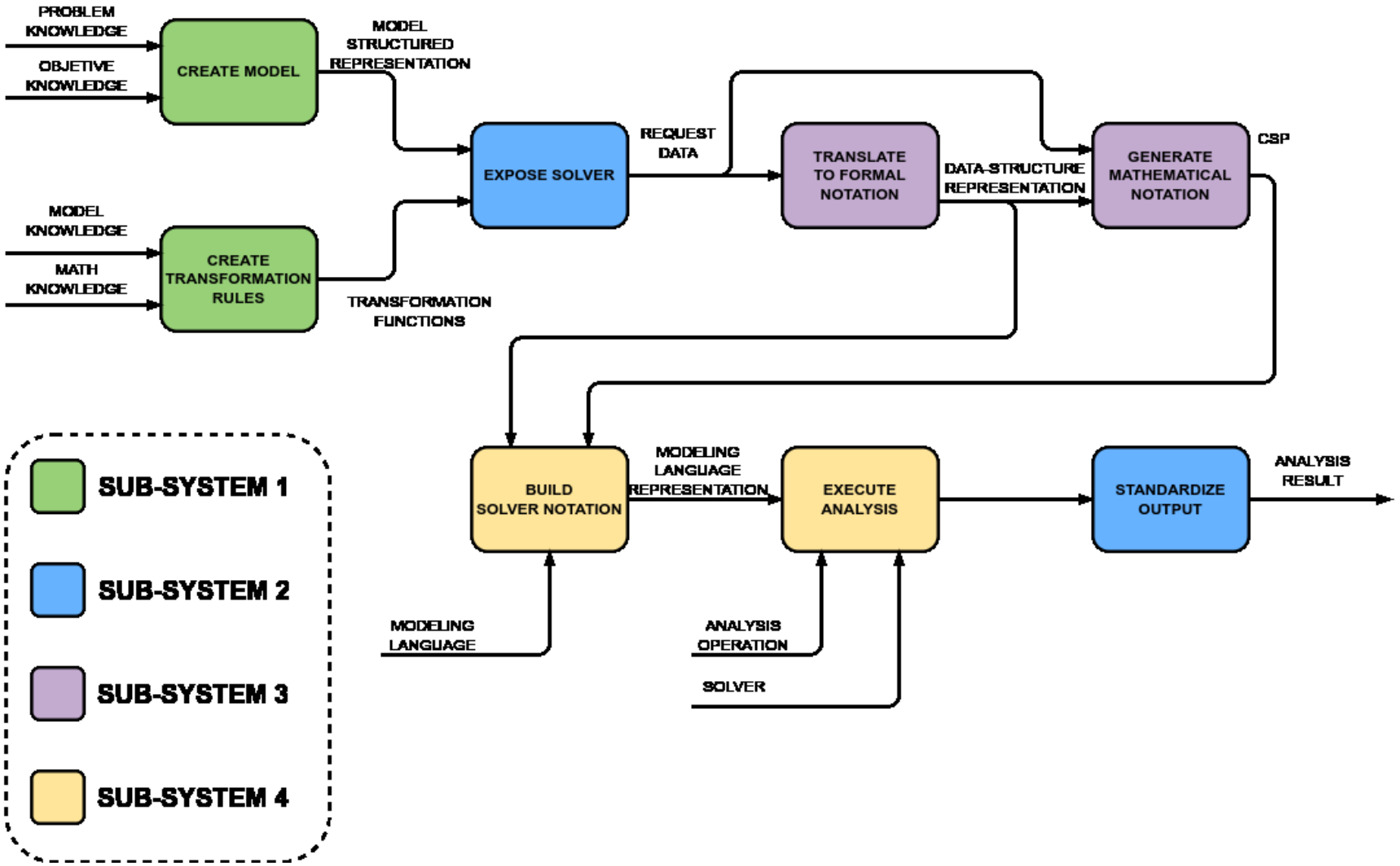

FIGURA 4.2: *Estructura funcional de backend de solvers genérico.*

4.2.5.1 - SUBSISTEMA 1

Este subsistema es la interfaz principal entre el usuario y el sistema. Es el lugar inicial donde se interactúa con el sistema y se desarrolla el modelo deseado. El usuario especifica el modelo mediante una representación gráfica, que es una forma visual de modelar los distintos componentes y relaciones dentro del modelo.

Una vez que el usuario ha especificado el modelo mediante la representación gráfica, el subsistema establece reglas de mapeo para transformar el modelo en un problema de satisfacción de restricciones. Las reglas de mapeo definidas en este subsistema toman la representación gráfica del modelo como entrada y la utilizan para generar las variables y restricciones del CSP. Esto permite que el sistema traduzca automáticamente el modelo especificado por el usuario a una forma que se pueda utilizar para resolver el CSP. El CSP resultante se puede enviar luego a otros subsistemas dentro del sistema para su procesamiento y soluciones adicionales.

4.2.5.2 - SUBSISTEMA 2

Este subsistema sirve como fachada, o interfaz, para los demás componentes del sistema. Su principal propósito es permitir que los usuarios interactúen con los solvers del sistema y reciban la salida de una manera fácil de interpretar. Este subsistema no realiza ninguna transformación significativa en los datos, pero es esencial para habilitar a los clientes externos a interactuar con el sistema de una manera estandarizada.

Para facilitar que los usuarios consuman las funciones de análisis proporcionadas por el sistema, es necesario asegurar que la salida del sistema sea uniforme y consistente. El subsistema es responsable de este proceso, transformando las salidas de diferentes

solvers en una sola entidad que pueda ser fácilmente comprendida y administrada por los clientes del sistema.

El subsistema actúa como puente entre el usuario y los solvers, facilitando la comunicación y asegurando que la salida se presente en un formato claro y comprensible. Esto es especialmente importante cuando el sistema es utilizado por clientes que pueden no tener conocimientos previos en los detalles técnicos de los solvers o los algoritmos subyacentes. Al proporcionar una interfaz sencilla e intuitiva, el subsistema facilita a los usuarios obtener los resultados que necesitan sin tener que entender la complejidad de los procesos subyacentes.

Además, este subsistema se encarga de asegurar la uniformidad y consistencia de la salida del sistema, transformando las salidas de diferentes solvers en una sola entidad que pueda ser fácilmente comprendida y administrada por los clientes. Esto asegura que el sistema sea fácil de usar y administrar, con una salida intuitiva y amigable para el usuario.

Este proceso es importante por varias razones. Primero, asegura que la salida del sistema sea fácil de entender e interpretar, independientemente del solver que se haya utilizado para generar la salida. Esto hace posible que los usuarios obtengan los resultados que necesitan sin tener que entender los detalles técnicos de los solvers. En segundo lugar, la uniformidad de la salida ayuda a asegurar que el sistema sea fácil de usar y administrar. Al presentar la salida en un formato consistente, el sistema se vuelve más intuitivo y amigable para el usuario, lo que facilita a los clientes trabajar con el sistema y obtener los resultados que necesitan.

En general, el papel de este subsistema es habilitar a los usuarios a interactuar con el sistema de manera ágil y sencilla, sin requerir que tengan un profundo conocimiento de

los detalles técnicos de los solvers o los procesos de transformación de datos. También, el subsistema se asegura de que la salida del sistema sea uniforme y fácil de entender, transformando las salidas de diferentes solvers en una sola entidad que pueda ser fácilmente consumida por los clientes del sistema. Este es un componente importante ya que ayuda a hacer que el sistema sea más fácil de usar y más intuitivo para los clientes.

4.2.5.3 - SUBSISTEMA 3

Al analizar modelos, es necesario transformarlos en una estructura que sea comprensible para los solvers. Este puede ser un proceso complejo, especialmente cuando los modelos se representan utilizando una variedad de formatos diferentes y las reglas de transformación no están normalizadas. En tales casos, puede ser casi imposible transformar efectivamente los modelos en una forma que pueda ser utilizada por los solvers.

Con el fin de superar este desafío, el subsistema 3 está diseñado para convertir la salida del subsistema 2 en una estructura de datos o formato normalizado. Este proceso es esencial para habilitar el razonamiento con los solvers, ya que permite que los modelos se presenten de una manera que sea consistente y fácil de comprender.

Al normalizar la estructura de datos o formato utilizado para representar los modelos, el subsistema 3 ayuda a asegurar que el proceso de transformación sea lo más eficiente y efectivo posible. Esto permite analizar una amplia gama de modelos diferentes, independientemente de su formato o representación original.

En general, el papel del subsistema 3 es facilitar la transformación de los modelos en una forma que sea utilizable por los solvers, convirtiendo la salida del subsistema 2 en una

estructura de datos o formato normalizado. Este es un paso crucial en el proceso de analizar modelos utilizando el sistema, ya que asegura que los solvers puedan razonar efectivamente sobre los modelos y generar soluciones precisas.

4.2.5.4 - SUBSISTEMA 4

Para analizar y recopilar información de un modelo, es necesario representarlo de una manera específica para el solver que se esté utilizando. El subsistema 4 está diseñado para manejar este proceso proporcionando un conjunto de funciones que pueden transformar la entrada en diferentes lenguajes de destino. Esta traducción es un paso crucial para habilitar al sistema a trabajar con múltiples solvers, ya que permite que el modelo sea representado de una manera compatible con una variedad de diferentes solvers. Adicionalmente, es responsable de ejecutar las operaciones de análisis sobre los modelos. El proceso de análisis es el corazón del sistema completo, ya que permite que el sistema valide la estructura y lógica de los modelos y recopile información importante sobre su comportamiento.

El uso de múltiples lenguajes de destino es un aspecto importante del subsistema, ya que permite al sistema lograr una mayor cobertura de solvers. Al no depender de un único lenguaje de destino como MiniZinc, el sistema es capaz de soportar una amplia gama de solvers y proporcionar más flexibilidad en términos de los solvers que se pueden utilizar. El hecho de utilizar exclusivamente MiniZinc excluye solvers muy importantes en herramientas de PLE, como SAT4J y JavaBDD.

Tener datos que puedan validar la estructura y lógica de los modelos es esencial para habilitar eficiencias en la ingeniería de líneas de productos a escala. Al permitir que diferentes análisis se ejecuten en varios solvers y paradigmas, el sistema se convierte en

una poderosa herramienta de experimentación e investigación para la ingeniería de líneas de productos. Esto hace posible probar y evaluar el comportamiento de los modelos utilizando una amplia gama de solvers y enfoques diferentes, lo que ayuda a identificar cualquier problema potencial o áreas de mejora.

El papel de este subsistema es facilitar la transformación de la entrada en una representación específica para el solver, utilizando un conjunto de funciones que pueden traducir la entrada a diferentes lenguajes de destino. Este es un paso importante en el proceso de analizar y recopilar información del modelo, y permite al sistema trabajar con una amplia gama de solvers y proporcionar más flexibilidad en términos de los solvers que se pueden utilizar. Además, se encarga de ejecutar las operaciones de análisis sobre los modelos, utilizando una variedad de solvers y paradigmas para recopilar información importante sobre los modelos para validar su estructura y lógica. Este es un componente esencial del sistema, ya que lo habilita para trabajar de manera eficiente y efectiva, y ayuda a convertirlo en una **útil herramienta** de experimentación e investigación para la ingeniería de líneas de productos.

4.2 - DISEÑO CONCEPTUAL

El diseño conceptual es una fase crítica en el proceso de ingeniería y diseño de productos. En esta etapa, se toma una definición del problema y se estructura un conjunto de posibles soluciones, que son amplias en foco y requieren una definición a profundidad. El diseñador tiene un papel clave en esta fase, ya que sus decisiones y cambios tienen un gran impacto en el resultado final del producto.

La importancia del diseño conceptual radica en que es el punto de partida para el desarrollo del producto. Es en esta fase donde se exploran las posibles soluciones y se

toman decisiones importantes sobre la arquitectura del producto. **[30]** Una vez que se han definido las soluciones conceptuales, se pueden evaluar sus ventajas y desventajas y elegir la que mejor cumpla con los requisitos del cliente y los objetivos del proyecto.

En la presente sección, se procederá a definir las posibles propuestas que permitan llevar a cabo la implementación del sistema de solvers genérico. Para ello, se emplea tanto la información proveniente del estado del arte como los requisitos establecidos para el sistema, con el objetivo de plasmar un conjunto de alternativas viables para la resolución del problema planteado.

4.2.1 - MATRIZ MORFOLÓGICA

La matriz morfológica es una herramienta metodológica que se utiliza en diversos campos del conocimiento, especialmente en la ingeniería y el diseño, para generar soluciones creativas a problemas complejos. Esta técnica combinatoria permite la exploración sistemática de múltiples combinaciones de elementos funcionales que conforman un sistema, con el fin de encontrar nuevas y diversas variantes de solución al problema en cuestión.

En términos prácticos, la matriz morfológica se representa como una tabla que consta de múltiples columnas y filas. En las columnas se identifican los elementos funcionales del sistema a analizar, mientras que en las filas se describen las diferentes variantes que se pueden combinar. **[28]** Al cruzar las filas y columnas de la tabla, se generan múltiples combinaciones que permiten explorar las posibles soluciones al problema.

SUBSISTEMA	OPCIONES					
Modelado	JSON		YAML		XML	
Servidor	HTTP		RPC		ASYNC	
Estandarización	CLIF	DSL	MiniZinc	Graph		
Solver	SAT4J	CHOCO	Prolog	JavBDD	Z3	JaCoP
	Alloy	MiniSAT	Maude	GLPK	Pothon	PicoSAT
Analizis	Internal DSL	External DSL	Query Lang	MiniZinc		
Regularización	JSON		YAML		XML	

TABLA 4.16: *Matriz morfológica del sistema de solvers*

4.2.2 - ALTERNATIVAS DE SOLUCIÓN

En esta sección, nos enfocaremos en desarrollar tablas comparativas detalladas que ilustren los diferentes subsistemas y los componentes potenciales que podrían utilizarse para construirlos. Nuestro objetivo principal es proporcionar una visión clara y comprensible de las múltiples opciones viables de implementación para el sistema en cuestión.

Para lograr esto, examinaremos cada subsistema y los posibles componentes que le conformarían. Una vez que hayamos presentado todas las opciones viables de implementación, procederemos a comparar y analizar cuidadosamente cada una de ellas. Este proceso de análisis nos permitirá determinar cuál es la solución más adecuada según los requisitos y las necesidades específicas del sistema que buscamos desarrollar.

SUBSISTEMA	OPCIONES					
Modelado	JSON		YAML		XML	
Servidor	HTTP		RPC		ASYNC	
Estandarización	CLIF		DSL		MiniZinc	Graph
Solver	SAT4J	CHOCO	Prolog	JavBDD	Z3	JaCoP
	Alloy	MiniSAT	Maude	GLPK	Pothon	PicoSAT
Analizis	Internal DSL		External DSL		Query Lang	MiniZinc
Regularización	JSON		YAML		XML	

TABLA 4.17: *Alternativa de solución 1*

SUBSISTEMA	OPCIONES					
Modelado	JSON		YAML		XML	
Servidor	HTTP		RPC		ASYNC	
Estandarización	CLIF		DSL		MiniZinc	Graph
Solver	SAT4J	CHOCO	Prolog	JavBDD	Z3	JaCoP
	Alloy	MiniSAT	Maude	GLPK	Pothon	PicoSAT
Analizis	Internal DSL		External DSL		Query Lang	MiniZinc
Regularización	JSON		YAML		XML	

TABLA 4.18: *Alternativa de solución 2*

SUBSISTEMA	OPCIONES					
Modelado	JSON		YAML		XML	
Servidor	HTTP		RPC		ASYNC	
Estandarización	CLIF		DSL		MiniZinc	Graph
Solver	SAT4J	CHOCO	Prolog	JavBDD	Z3	JaCoP
	Alloy	MiniSAT	Maude	GLPK	Pothon	PicoSAT
Analizis	Internal DSL		External DSL		Query Lang	MiniZinc
Regularización	JSON		YAML		XML	

TABLA 4.19: *Alternativa de solución 3*

SUBSISTEMA	OPCIONES					
Modelado	JSON		YAML		XML	
Servidor	HTTP		RPC		ASYNC	
Estandarización	CLIF		DSL		MiniZinc	Graph
Solver	SAT4J	CHOCO	Prolog	JavBDD	Z3	JaCoP
	Alloy	MiniSAT	Maude	GLPK	Pothon	PicoSAT
Analizis	Internal DSL		External DSL		Query Lang	MiniZinc
Regularización	JSON		YAML		XML	

TABLA 4.20: *Alternativa de solución 4*

4.3 - REFINAMIENTO DE LOS ESQUEMAS

Durante la etapa de refinamiento de los esquemas, se examinan meticulosamente las múltiples soluciones propuestas previamente en la fase de diseño conceptual, con la intención de identificar y seleccionar una de ellas. Este proceso de selección se lleva a cabo mediante la comparación de las diversas opciones en función de las métricas de calidad previamente establecidas para el sistema en cuestión. Dichas métricas facilitan la medición y evaluación del rendimiento de las alternativas de solución con respecto a su habilidad para satisfacer los requisitos funcionales y no funcionales previamente definidos.

Es importante destacar que en algunos casos, ciertas métricas de los requisitos no funcionales no pueden ser calculadas de manera precisa antes de que el producto sea desarrollado. Esto se debe a que estas métricas están relacionadas con aspectos del sistema que no pueden ser completamente comprendidos hasta que se implementen y se ejecuten en un entorno real.

Ante esta situación, es común utilizar proxies de estas métricas para evaluar las alternativas de solución durante la fase de refinamiento. Un proxy es una medida o indicador que se utiliza como sustituto de otra métrica que no se puede calcular directamente.

Es importante destacar que el proceso de refinamiento implica una evaluación exhaustiva de cada alternativa de solución, teniendo en cuenta su viabilidad técnica y su factibilidad, se deben considerar no sólo las métricas de los requisitos no funcionales, sino también

las limitaciones y restricciones del entorno y contexto en el que se desenvolverá el sistema.

4.3.1 - PARÁMETROS DE SELECCIÓN

Para poder llevar a cabo una evaluación exhaustiva de las alternativas de solución propuestas, es fundamental establecer una metodología rigurosa y estructurada que permita describir y evaluar las métricas relevantes para cada uno de los subsistemas implicados en el problema. Para ello, es necesario definir una matriz de evaluación que permita identificar y medir las diferentes métricas y valores de significancia correspondientes a cada sub-sistema involucrado.

La matriz de evaluación es una herramienta que permite estructurar y organizar las diferentes métricas y valores de significancia que serán utilizados para evaluar cada sub-sistema. **[30]**

PARAMETRO	VALOR	COMPONENTE			
		SERIALIZACIÓN			
		JSON	YAML	XML	
M-4	10%	4	4	2	
M-6	10%	5	3	2	
		SERVIDOR			
		HTTP	RPC	ASYNC	
M-1	10%	4	5	5	
M-4	5%	4	3	2	
M-5	5%	5	4	1	
		ESTANDARIZACIÓN			
		CLIF	DSL	MiniZinc	Graph
M-3	10%	5	5	2	2

M-4	5%	4		4		4	5
		SOLVER					
		SAT4J	CHOCO	Prolog	JavaBDD	Z3	JaCoP
M-3	25%	5	5	3	4	2	2
M-4	5%	2	5	3	5	3	2
		Alloy	MiniSAT	Maude	GLPK	Photon	PicoSAT
M-3	25%	1	1	1	1	1	1
M-4	5%	3	1	2	4	3	1
		ANALIZIS					
		Internal DSL		External DSL		Query Lang	MiniZinc
M-1	15%	5		3		2	2

TABLA 4.21: *Matriz valores de evaluación*

4.3.3 - SELECCIÓN

Con el propósito de elegir la solución más adecuada, buscamos realizar una evaluación cuantitativa de cada una de las propuestas. Mediante la tabla expuesta en la sección anterior, podemos asignar un valor numérico a cada elemento y determinar cómo estos contribuyen a una evaluación integral del sistema.

La siguiente tabla representa una evaluación cuantitativa de las cuatro soluciones propuestas en base a los parámetros de la tabla de parámetros de selección. Cada parámetro tiene asignado un porcentaje de importancia y un puntaje para cada solución. Al final, se calcula un total ponderado para cada solución.

PARAMETRO	VALOR	SOL. 1	SOL. 2	SOL. 3	SOL. 4
Serialización - [M-4]	10%	2	4	4	4
Serialización - [M-6]	10%	2	5	3	5
Servidor - [M-1]	10%	4	5	5	4
Servidor - [M-4]	5%	4	3	2	4
Servidor - [M-5]	5%	5	4	1	5
Estandarización - [M-3]	10%	2	5	2	5
Estandarización - [M-4]	5%	4	4	5	4
Solver - [M-3]	25%	0.55	1.85	0.15	3.51
Solver - [M-4]	5%	0.55	1.34	0.2	3.32

Analizis - [M-1]	15%	2	5	2	5
TOTAL		**2.12**	**3.73**	**2.14**	**4.24**

TABLA 4.22: *Matriz para la selección de la propuesta*

Esta tabla permite comparar las soluciones de manera numérica y ponderada. Según los resultados, la **Solución 4 obtiene el mayor puntaje total (4.24)**, lo que indica que es la opción más favorable basándonos en los parámetros y valores asignados.

4.4 - DETALLADO

La fase de detallado se enfoca en la especificación de un producto particular resultante de las decisiones tomadas en las etapas previas del proceso de diseño. Esta etapa es crucial para la materialización del concepto, ya que involucra la generación de modelos detallados y modelos precisos para garantizar que el producto cumpla con las expectativas y requisitos identificados en fases anteriores del proceso de diseño.

En esta sección, abordaremos en detalle la propuesta específica que hemos desarrollado para el sistema genérico de solvers. A través de un análisis exhaustivo de cada uno de sus componentes, proporcionaremos una comprensión profunda de la estructura y funcionamiento del sistema propuesto. El objetivo de esta sección es desglosar cada elemento del sistema de solvers, analizando su función, características y la manera en que interactúa con los demás componentes. Esta descripción detallada permitirá evaluar cómo el sistema particular se adapta a los requisitos previamente descritos, proporcionando una base sólida para el análisis de su viabilidad y aplicabilidad en diferentes contextos.

4.4.1 - REPRESENTACIÓN GRÁFICA

La evolución del desarrollo de software y sistemas pone de manifiesto el papel crucial que desempeña la abstracción en el manejo de la complejidad. Un enfoque efectivo para gestionar dicha complejidad en una línea de productos, ésta consiste en extraer los requisitos comunes y variables de un conjunto indeterminado de productos y organizarlos en un modelo.

Estos modelos, representados gráficamente, no solo abstraen la variabilidad y las similitudes de los productos dentro de una línea específica, sino que también pueden emplearse para optimizar los procesos de toma de decisiones y mejorar la comunicación entre los participantes en la ingeniería y operación de la línea de productos.

En el marco del presente documento, seleccionaremos a VariaMos como nuestra interfaz de usuario principal debido a sus múltiples ventajas y características en el ámbito del modelado. VariaMos proporciona a sus usuarios un entorno integrado y flexible en el cual es posible definir diversos modelos, así como sus respectivas representaciones gráficas. Este enfoque permite a los usuarios personalizar el modelado según sus necesidades específicas y objetivos, facilitando la creación de soluciones adaptadas a contextos particulares.

Una de las principales fortalezas de VariaMos radica en su capacidad para posibilitar la incorporación de modelos personalizados de manera sencilla y eficiente. Los usuarios pueden diseñar y añadir sus propios modelos, definiendo tanto la estructura subyacente como la representación gráfica, lo que otorga mayor autonomía y adaptabilidad en función de los requisitos de cada proyecto.

4.4.1 - REPRESENTACIÓN SEMI-ESTRUCTURADA

Los lenguajes semi-estructurados desempeñan un papel fundamental en el intercambio de información entre aplicaciones de manera simplificada y eficiente. La utilidad de estos lenguajes radica en su capacidad para representar la información contenida en modelos gráficos de tal forma que los sistemas informáticos puedan procesar y operar sobre dicha información. Aunque estos lenguajes no permiten realizar operaciones matemáticas directamente, facilitan el envío de información de cualquier modelo a un software especializado que pueda analizar la validez y consistencia del modelo creado.

VariaMos, en su calidad de herramienta de modelado, incorpora una interfaz de lenguaje semi-estructurado basada en JSON. Esta interfaz posibilita la representación de distintos modelos en diversas formas. La implementación de esta interfaz en JSON ofrece ventajas en términos de compatibilidad con diferentes plataformas y sistemas.

La interfaz en cuestión opera mediante la construcción de un grafo, el cual consta de nodos y conexiones. Los nodos representan los elementos fundamentales del modelo, mientras que las conexiones simbolizan las relaciones entre dichos elementos. Esta estructura de grafo facilita la representación de modelos gráficos de manera intuitiva y coherente, al tiempo que permite la realización de análisis y operaciones sobre los datos contenidos en el modelo.

```
{
  "type": "object",
  "properties": {
    "id": { "type": "string" },
    "name": { "type": "string" },
    "elements": {
```

```
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "id": { "type": "string" },
          "type": { "type": "string" },
          "name": { "type": "string" },
          "properties": { "type": "array",
            "items": {
              "type": "object",
              "properties": {
                "id": { "type": "string" },
                "name": { "type": "string" },
                "type": { "type": "string" },
              "required": ["id", "name", "type"]
            }
          }
        },
        "required": ["id", "type", "name", "properties"]
      }
    },
    "relationships": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "id": { "type": "string" },
          "type": { "type": "string" },
          "name": { "type": "string" },
          "sourceId": { "type": "string" },
          "targetId": { "type": "string" },
          "properties": { "type": "array" }
        },
        "required": [
          "id", "type", "name", "sourceId", "targetId"
        ]
```

```
      }
    }
  },
  "required": ["id", "name", "elements", "relationships"]
}
```

El esquema JSON presentado anteriormente define un objeto que representa una estructura compleja compuesta por elementos y relaciones entre ellos. El objeto principal requiere propiedades de identificación y nombre, así como una lista de elementos y relaciones. Cada elemento incluye un identificador, tipo, nombre y un conjunto de propiedades asociadas, mientras que las relaciones incluyen un identificador, tipo, nombre y los identificadores de los objetos origen y destino, además de un conjunto de propiedades. La descripción detallada de este esquema revela una estructura jerárquica y anidada que permite modelar objetos y sus interrelaciones de manera flexible y extensible.

4.4.2 - REPRESENTACIÓN EN ESTRUCTURA DE DATOS

Los modelos de líneas de productos generalmente están compuestos de elementos conectados entre sí por diferentes tipos de relación. Un elemento importante de estas conexiones es que también pueden tener precedencia y esta puede ser significativa para el modelo. Un ejemplo de dichas relaciones existe en los modelos de características cuando se encuentra una subordinación de característica padre a característica hija.

Se hace claro que una forma simple de entender los modelos es como un grafo, en el que tenemos nodos y arcos que los unen entre ellos. Es importante denotar que existen diferentes tipos de grafos, y que es pertinente seleccionar el que más se adapte a las necesidades planteadas por los modelos utilizados para líneas de producto.

Como primera aproximación a este tipo de modelos, una conjetura informada del tipo de grafo que estamos tratando podría ser un grafo dirigido. Este aparenta ser un buen candidato ya que explica la conexión entre los nodos del modelo y la precedencia jerárquica existente en estos.

Un grafo dirigido G es un grafo en que los arcos tienen una dirección. Este puede ser representado por una tupla $G = (V, E)$ donde V es un conjunto de todos los nodos que existen en el grafo, y E es otro conjunto que contiene tuplas representando las conexiones entre los nodos, donde la primer componente de la tupla es el nodo inicial de la relación y la segunda componente es el nodo de destino. De manera formal podemos expresar lo anterior como estas dos condiciones:

- $V \neq \emptyset$
- $E \subseteq \{(x, y) \mid (x, y) \in V^2\}$

A pesar de que este tipo de grafo puede describir la mayoría de los modelos que utilizamos en VariaMos, como los “Feature Models”, “IStar”, “REFAS” y “ADL” adolece de algunas falencias que impiden describir otros tipos de modelo. La primera y más grande falencia viene del hecho de que los arcos en este caso están tipados, esto nos confirma que nada impide a un nodo unirse a otro con dos arcos idénticos pero tipados de forma distinta. Esto es problemático ya que en el conjunto E repetir valores de la tupla (x, y) no tiene sentido, ya que en un conjunto los valores repetidos se interpretan como si fuesen el mismo valor.

Para ilustrar el problema previamente descrito podemos utilizar un “String Diagram”.

Supongamos que existe una función que no recibe entradas pero genera dos salidas con tipos diferentes, y una segunda función que recibe dos parámetros como entrada y genera una salida de un tercer tipo distinto, esto puede ser ilustrado como muestra la siguiente imagen (Figura 3).

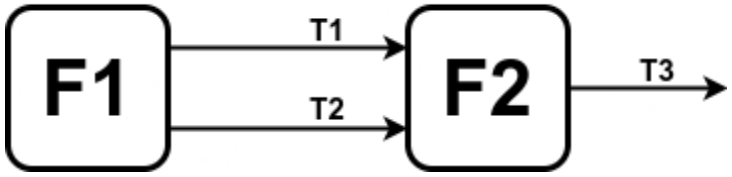

FIGURA 4.3: *String model ejemplificando conexiones tipadas.*

Si queremos expresar el modelo como un grafo dirigido es necesario definir los conjuntos E y V esto lo podemos hacer de la siguiente forma:

- $V = \{F1, F2\}$
- $E = \{(F1, F2), (F1, F2), (F2, \emptyset)\}$

Como podemos ver el problema se manifiesta claramente en la primera y segunda tupla del conjunto E que al estar en un conjunto se infieren como el mismo arco a pesar de que su tipado diferente sugiere que al interpretar el modelo estas relaciones tienen significados totalmente distintos.

Para poder solucionar este inconveniente es importante introducir el concepto de multigrafo dirigido. En primer lugar un multigrafo es una generalización de los grafos que permite arcos múltiples o paralelos, o más de un conjunto de arcos. De esta forma solucionamos el hecho de tener dos nodos conectados por más de una arista.

Podemos definir matemáticamente un multigrafo dirigido Γ como una 4-tupla $\Gamma = (E, V, s, t)$ donde V es el conjunto de nodos que componen el multigrafo, E es un conjunto de etiquetas que representan un arco, s es una función $s: V \to E$ con un mapa que dada una etiqueta de arco retorna el vértice de origen de la relación, y t es una función $t: V \to E$ con un mapa que dada una etiqueta de arco retorna el vértice de destino de la relación. **[31]**

Si volvemos al ejemplo del "String Diagram" podemos expresarlo utilizando un multigrafo dirigido así:

- $V = \{F1, F2\}$
- $E = \{T_1, T_2, T_3\}$

Como se hace evidente de las ecuaciones previas, el hecho de no depender de tuplas en el conjunto E y en reemplazo nombrar cada arco de forma única con un label para poderlo utilizar con las funciones t y s hace posible tener más de un arco que compartan nodos de origen y destino.

Ya que nuestro objetivo es generalizar y tener una única representación de los modelos, es pertinente aclarar que los $t(x) = \begin{cases} F2 & si \quad x = T_1 \\ F2 & si \quad x = T_2 \\ \emptyset & si \quad x = T_3 \end{cases}$ leben $s(x) = \begin{cases} F1 & si \quad x = T_1 \\ F1 & si \quad x = T_2 \\ F2 & si \quad x = T_3 \end{cases}$ poseen conexiones tipadas tipo productos. Finalmente todo grafo dirigido puede ser representado como un multigrafo dirigido, un grafo dirigido es un caso especial de un multigrafo dirigido.

Para poder describir lo enunciado anteriormente de forma matemática pensemos en un

conjunto que contiene todos los posibles grafos dirigidos DAG y otro conjunto que contiene todos los posible multigrafos dirigidos $MDAG$. Entonces podemos decir que todo elemento existente en DAG se encuentra en $MDAG$, pero hay elementos de $MDAG$ que no pueden existir en DAG, lo que implica consecuentemente que $DAG \subset MDAG$.

Es importante aclarar que algunos de los modelos de líneas de productos tienen estructura de árbol. Estos pueden ser entendidos como un grafo dirigido sin ciclos, es por esta razón que si tenemos un conjuntos A de todos los posibles árboles, es válido decir que $A \subset DAG$ y por transitividad llegamos al hecho de que $A \subset MDAG$ por lo que podemos asegurar que también todo modelo en forma de árbol puede ser descrito por un multigrafo dirigido.

En esencia, el propósito subyacente de esta estructura de datos consiste en proporcionar una representación análoga a la de los lenguajes semi-estructurados, pero con el beneficio de ser fácilmente manipulable desde cualquier lenguaje de programación. En el contexto específico de nuestro estudio, deseamos implementar dicha estructura de datos en el lenguaje de programación Python.

```
class Node:
    def __init__(self, **kwargs: dict[str, typing.Any]) -> None:
        self.validate_id(kwargs)
        self.parameters = kwargs

class Connection:
    def __init__(self, **kwargs: dict[str, typing.Any]) -> None:
        self.validate_id(kwargs)
        self.validate_source(kwargs)
        self.validate_destinations(kwargs)
        self.parameters = kwargs
```

```
        self.parameters["destination"] = self.parameters[
            "Destinations"
        ][0]

class MGraph:
    def __init__(
        self,
        nodes: list[Node] = None,
        connections: list[Connection] = None
    ) -> None:
        self.nodes = nodes if nodes else []
        self.connections = connections if connections else []
        self._node_table = {
            node.id: node for node in self.nodes
        }
        self._connection_table = {
            connection.id: connection
            for connection in self.connections
        }

        self._sources_table = {}
        self._destinations_table = {}

        for connection in self.connections:
            self.add_connection(connection)

    def add_node(self, node: Node) -> None:
        ...

    def add_connection(self, connection: Connection) -> None:
        ...
```

El código proporcionado define tres clases en Python: Node, Connection y MGraph. La clase Node representa un nodo en un grafo y contiene un constructor que valida su identificador y almacena sus parámetros en un diccionario. La clase Connection

representa una conexión entre nodos y contiene un constructor que valida su identificador, su nodo de origen y sus nodos de destino, además de almacenar sus parámetros en un diccionario. La clase MGraph representa un multigrafo dirigido que contiene nodos y conexiones. El constructor de la clase acepta dos listas de objetos (Node y Connection) como argumentos, y crea tablas internas para mantener un registro de los nodos y conexiones por sus identificadores. La clase MGraph proporciona métodos para agregar nodos y conexiones al grafo, y para obtener nodos y conexiones por sus identificadores, así como para obtener conexiones dado el identificador de un nodo de origen o destino.

Es preciso puntualizar que, en esta implementación específica de multigrafo, se asume que cada instancia perteneciente a la clase Connection posee un tipado diferente. Este enfoque facilita que múltiples relaciones, las cuales conceptualmente exhiben el mismo tipo, puedan compartir puntos de origen y destino idénticos. Un ejemplo paradigmático de esta situación se presenta en un modelo de características que contiene diversas relaciones de tipo OR o AND entre características compartidas.

4.4.2.1 - RAZONAMIENTO ESTRUCTURAL

Una particularidad innovadora de la presente propuesta radica en su capacidad para llevar a cabo razonamientos estructurales sobre los modelos, habilidad ausente en los trabajos previamente mencionados en la literatura. Las operaciones de razonamiento estructurales pueden definirse como aquellas que operan sobre la estructura de grafo previamente establecida. Gracias a estas operaciones, es factible verificar atributos cuya comprobación resultaría compleja mediante el empleo de un solver CSP.

Una característica relevante que podemos verificar en los modelos mediante este tipo de operaciones es la conexidad del grafo. Esto implica que siempre existe una ruta dentro del grafo que permite acceder a cualquier otro nodo arbitrario de la estructura. La importancia de esta propiedad radica en que los grafos disconexos pueden denotar un modelo incompleto o incorrectamente planteado. En el siguiente código podemos ver como se puede realizar esta operación.

```
def is_connected(graph: MGraph) -> bool:
    visited = set()
    start_node = graph.nodes[0]

    queue = deque([start_node])

    while queue:
        current_node = queue.popleft()
        visited.add(current_node)

        connections = (
            graph.get_connections_from_source(current_node.id)
        )

        if connections:
            for connection in connections:
                for destination in connection.destinations:
                    if destination not in visited:
                        queue.append(destination)

    return len(visited) == len(graph.nodes)
```

La función "is_connected" verifica si un grafo, representado por una instancia de la clase "MGraph", está conectado utilizando una búsqueda en amplitud (BFS). Comienza con el primer nodo del grafo, explora sus vecinos y los agrega a una cola para explorarlos en

iteraciones futuras. El proceso se repite hasta que todos los nodos hayan sido visitados o la cola esté vacía. Al final, si el número de nodos visitados es igual al número total de nodos en el grafo, se considera que el grafo está conectado y la función devuelve "True", de lo contrario, devuelve "False".

Una cualidad destacable de estas operaciones es su idoneidad como candidatas para la composición funcional. Esto alude a la capacidad de dichas funciones para ser aplicadas sucesivamente sobre la misma entrada en distintas combinaciones, generando nuevas operaciones. Un ejemplo ilustrativo de esta situación es una función que determina si un grafo posee una estructura de árbol; para ello, se emplean dos operaciones previamente definidas, tales como "is_connected" y "has_cycle".

```
def is_tree(graph: MGraph) -> bool:
    if not is_connected(graph):
        return False

    return not has_cycle(graph)
```

Podemos ver como solo con estas dos funciones previamente definidas podemos obtener una nueva operación importante para los modelos de líneas de producto. Particularmente esta operación es importante en el contexto de los modelos de características, ya que si quitamos las relaciones de "excluye" e "incluye" el modelo debe tener estructura de árbol.

4.4.3 - REPRESENTACIÓN MATEMÁTICA

Para llevar a cabo las operaciones convencionales presentes en la literatura de líneas de productos, es imprescindible transformar el modelo que debemos evaluar en un

Problema de Satisfacción de Restricciones (CSP, por sus siglas en inglés). Para lograr esto, desarrollamos un lenguaje de dominio específico que facilita la expresión de problemas de satisfacción de restricciones.

La gramática puede generar cuatro tipos de restricciones: lógicas, relacionales, aritméticas y de variables. Estas restricciones son útiles para modelar y representar diversas condiciones y relaciones en problemas de satisfacción de restricciones (CSP), permitiendo expresar de manera precisa las limitaciones y dependencias entre variables, así como las operaciones permitidas en el dominio del problema.

Previo a abordar las restricciones que esta gramática puede evaluar, es fundamental discutir los parámetros y variables posibles que pueden ser empleados en dichas restricciones. El primer tipo de variable de interés son las entidades, las cuales están vinculadas al grafo que representa el problema. Esto permite expresar restricciones en función de los elementos del multigrafo dirigido, facilitando así la representación de las condiciones y limitaciones del problema en términos de su estructura gráfica. Estas nos proporcionan dos elementos, "Node" y "Connection". Estos elementos nos permiten hacer referencia a los nodos y conexiones del multigrafo y sus propiedades. La definición de estas variables está representada en el siguiente código.

```
Value: type=EntityType "."  values+=ID["."];

EntityType: "Node"
          | "Connection";
```

Las variables en esta gramática se definen mediante la clase "Variable", que incluye un tipo específico de variable y una lista de parámetros asociados. Las variables pueden ser

de dos tipos: enteras ("Integer") o booleanas ("Boolean"). Estas variables sirven como elementos fundamentales para expresar restricciones y relaciones en un problema de satisfacción de restricciones (CSP) modelado con esta gramática. Los parámetros permitidos para las variables incluyen valores enteros, booleanos, entidades de valor y rangos. Podemos ver la definición de las variables de la gramática en el siguiente segmento de código.

```
Variable: type=VariableType "("
parameters+=VariableParameter[","] ")";

VariableParameter: INT | BOOL | Value | Range;

Range: "Range" "(" minimum=INT "," maximum=INT ")";

VariableType: "Integer"
              | "Boolean";
```

La restricción lógica en esta gramática se define mediante la clase "Logic", que consiste en un tipo específico de restricción lógica y una lista de parámetros. Los tipos de restricciones lógicas incluyen operadores como Equivalencia, Implicación, Negación, Conjunción (And), Disyunción (Or) y Disyunción exclusiva (Xor). Los parámetros permitidos para estas restricciones lógicas son valores booleanos, entidades de valor, así como otras restricciones lógicas o relacionales. Podemos ver como se define este tipo de restricción en la gramática en el siguiente código.

```
Logic: type=LogicType "(" parameters+=LogicParameter[","] ")";

LogicParameter: BOOL | Value | Logic | Relational;
```

```
LogicType: "Equivalence"
           | "Implication"
           | "Negation"
           | "And"
           | "Or"
           | "Xor";
```

La restricción relacional en esta gramática se define a través de la clase "Relational", que se compone de un tipo específico de restricción relacional y una lista de parámetros. Los tipos de restricciones relacionales abarcan operadores como Inferior, Superior, Igual y Diferente. Los parámetros permitidos en estas restricciones relacionales incluyen valores enteros, entidades de valor, inclusividad relacional y restricciones aritméticas. Las restricciones relacionales están definidas en la siguiente gramática.

```
Relational: type=RelationalType "("
parameters+=RelationalParameter[","] ")";

RelationalParameter: INT | Value | RelationalInclusiveness |
Arithmetic;

RelationalInclusiveness: "Inclusive" | "Exclusive";

RelationalType: "Inferior"
                | "Superior"
                | "Equal"
                | "Different";
```

La restricción aritmética en esta gramática se define mediante la clase "Arithmetic", que consiste en un tipo específico de restricción aritmética y una lista de parámetros. Los tipos de restricciones aritméticas incluyen operaciones como Adición, Sustracción, Multiplicación, División, Mínimo y Máximo. Los parámetros permitidos para estas

restricciones aritméticas son valores enteros, entidades de valor y otras restricciones aritméticas. Podemos ver la definición de las restricciones aritméticas en el siguiente código.

```
Arithmetic: type=ArithmeticType "("
parameters+=ArithmeticParameter[","] ")";

ArithmeticParameter: INT | Value | Arithmetic;

ArithmeticType: "Addition"
                | "Subtraction"
                | "Multiplication"
                | "Division"
                | "Minimum"
                | "Maximum";
```

El propósito principal de esta gramática es actuar como intermediario entre el usuario y diversos lenguajes de modelado. Específicamente, se trata de un lenguaje que se compilará en otros lenguajes de modelado objetivo. Algunos lectores podrían cuestionar si no es factible compilar directamente desde el multigrafo a los lenguajes de modelado. Si bien esto es posible, implicaría que el usuario se familiarizarse con el funcionamiento de cada uno de estos lenguajes. Por lo tanto, el lenguaje propuesto sirve como un único punto de conexión a estos lenguajes, simplificando el proceso y permitiendo a los usuarios utilizar una amplia gama de solvers sin la necesidad de ser expertos en ellos.

4.4.3.1 - REGLAS DE TRANSFORMACIÓN

En el ámbito de investigación de líneas de productos, un campo destacado se centra en encontrar representaciones más eficientes o con características específicas para modelos de líneas de productos. Con el fin de permitir a los usuarios explorar diversas maneras de

transformar el modelo en un CSP, optamos por crear un formato JSON que represente cómo se realizarán dichas transformaciones del multigrafo al CSP. Por lo general, en las herramientas de líneas de productos descritas en la literatura, el autor de la herramienta determina previamente la representación para cada modelo, lo que limita la eficacia de dichas herramientas en términos de investigación en PLE. En resumen, la presencia de estas reglas de transformación confiere flexibilidad a la herramienta y aumenta su utilidad en la investigación en ingeniería de líneas de productos.

```
{
  "type": "object",
  "properties": {
    "mapping": {
      "type": "object",
      "properties": {
        "nodes": {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "name": { "type": "string" },
              "variables": {
                "type": "array",
                "items": { "type": "string" }
              },
              "constraints": {
                "type": "array",
                "items": { "type": "string" }
              }
            },
            "required": ["name"]
          }
        },
        "connections": {
```

```
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "name": { "type": "string" },
              "variables": {
                "type": "array",
                "items": { "type": "string" }
              },
              "constraints": {
                "type": "array",
                "items": { "type": "string" }
              }
            },
            "required": ["name"]
          }
        }
      },
      "required": ["nodes", "connections"]
    }
  },
  "required": ["mapping"]
}
```

El esquema JSON proporcionado define un objeto principal con una propiedad requerida llamada "mapping", que es un objeto con dos propiedades requeridas: "nodes" y "connections", ambos de tipo "array". Cada elemento en "nodes" y "connections" es un objeto con tres propiedades requeridas: "name" (de tipo "string"), "variables" (un array de elementos "string") y "constraints" (un array de elementos "string"). El esquema representa una estructura de datos que incluye un mapeo con nodos y conexiones, cada uno con un nombre, variables y restricciones asociadas.

El siguiente esquema JSON muestra la forma de representar las diversas reglas de transformación para los elementos del multigrafo. La intención es que este mapeo se incluya junto con el formato semi-estructurado proporcionado por VariaMos, permitiendo así transformar el modelo en un CSP que pueda ser utilizado para realizar análisis del mismo.

4.4.3.2 - COMPILACIÓN

Nos referimos a la compilación como el proceso de traducción de un conjunto de restricciones, creadas utilizando la gramática del CSP previamente especificada, hacia un lenguaje de modelado objetivo. Con el propósito de permitir que el conjunto de lenguajes objetivo se expanda a lo largo del tiempo, hemos diseñado una interfaz que simplifica la implementación de nuevos compiladores para distintos lenguajes de modelado.

```
class Compiler:
    def __init__(self):
        pass

    def transpile_constraint(self, constraint):
        pass

    def transpile_logic(self, logic):
        pass

    def transpile_logic_parameter(self, logic_parameter):
        pass

    def transpile_relational(self, relational):
        pass

    def transpile_relational_parameter(self,
```

```
relational_parameter):
        pass

    def transpile_relational_inclusiveness(self,
relational_inclusiveness):
        pass

    def transpile_arithmetic(self, arithmetic):
        pass

    def transpile_arithmetic_parameter(self,
arithmetic_parameter):
        pass

    def transpile_variable(self, variable):
        pass

    def transpile_variable_parameter(self, variable_parameter):
        pass

    def transpile_range(self, range_obj):
        pass

    def transpile_value(self, value):
        pass

    def transpile_entity_type(self, entity_type):
        pass
```

La interfaz del compilador define una estructura común para traducir la gramática de CSP específica a diferentes lenguajes de modelado. La interfaz es útil porque permite encapsular la lógica de traducción en un conjunto de clases y métodos, lo que facilita el mantenimiento, la extensibilidad y la reutilización del código.

La interfaz del compilador proporciona una clase base llamada "Compiler", que contiene métodos para traducir cada elemento de la gramática. Estos métodos actúan como marcadores de posición y deben ser implementados en las clases especializadas para cada lenguaje de destino.

En el siguiente ejemplo, se crearán dos clases especializadas: "MiniZincCompiler" y "XCSP3Compiler". Estas clases heredan de la clase base "Compiler" y deben implementar los métodos de traducción específicos para cada lenguaje de destino.

```
class MiniZincCompiler(Compiler):
    # Implement the MiniZinc-specific translation methods here
    pass

class XCSP3Compiler(Transpiler):
    # Implement the XCSP-specific translation methods here
    pass
```

4.4.4 - LENGUAJES DE DESTINO

Los lenguajes de modelado matemático son los que nos interesan como lenguajes de destino en el marco de esta tesina. Esto debido a que permiten representar problemas y sistemas complejos de manera formal y estructurada. Estos lenguajes facilitan la comunicación entre expertos en distintas disciplinas y proporcionan un marco para analizar y resolver problemas mediante la formulación de modelos matemáticos. Al enfocarnos en estos lenguajes de modelado como lenguajes de destino, buscamos generar soluciones y optimizaciones para problemas del mundo real, al aprovechar algoritmos y técnicas matemáticas.

Los lenguajes de modelado matemático son herramientas especializadas para la elaboración y análisis de modelos matemáticos de sistemas reales, tales como sistemas físicos, biológicos, económicos y sociales. Estos lenguajes proporcionan una amplia variedad de herramientas para la manipulación de conceptos matemáticos y para la visualización y análisis de resultados de cálculos.

Una de las principales características de los lenguajes de modelado matemático es su expresividad, que les permite representar una amplia gama de conceptos matemáticos y ecuaciones, como restricciones, ecuaciones lineales, ecuaciones no lineales, problemas de optimización e incluso ecuaciones diferenciales. Estos lenguajes también se distinguen por ser altamente paralelos y tener un rendimiento excelente en la resolución de problemas altamente paralelizables e intensivos en cómputo.

En este trabajo de investigación, nos enfocamos en dos lenguajes de modelado matemático en particular: XCSP3 y MiniZinc. Ambos lenguajes son ampliamente utilizados en la comunidad científica y tienen características específicas que los hacen interesantes para nuestro estudio.

4.4.4.1 - MINIZINC

MiniZinc es un lenguaje de modelado matemático especialmente diseñado para expresar problemas de satisfacción y optimización. Fue desarrollado como parte del proyecto de investigación MINICP de la Universidad de Monash y es actualmente mantenido por un grupo de programadores dedicados a mejorarlo.

El propósito de MiniZinc es proporcionar una forma sencilla de expresar problemas de optimización y búsqueda, facilitar la utilización de una amplia variedad de solvers y técnicas de optimización para resolver los problemas escritos en este lenguaje. MiniZinc se ha diseñado para ser un lenguaje fácil de usar y entender, con una sintaxis concisa e intuitiva para la especificación de variables, restricciones y objetivos.

En términos generales, un programa en MiniZinc consta de tres conjuntos: el primero es el conjunto de variables, el segundo es el conjunto de restricciones y el tercero es el conjunto de relaciones entre variables. Además, el programa incluye una función objetivo que describe el criterio de optimización para el programa. Los modelos escritos en MiniZinc pueden ser resueltos utilizando una variedad de técnicas de razonamiento, como la programación lineal, la programación no lineal, la programación de enteros, la programación entera mixta y la satisfacción de restricciones.

MiniZinc también soporta la representación de una amplia gama de tipos de datos, incluyendo enteros, flotantes, booleanos, conjuntos, arreglos y tuplas. Además, admite una variedad de expresiones y estructuras de control, como las expresiones "let", "if-then-else", las cláusulas "case", los "forall" y las operaciones de existencia. **[32]** A continuación se presenta una versión simplificada de la gramática de MiniZinc en EBNF:

```
program = { (include | constraint | solve | output) }

include = "include" string

constraint = ( "constraint" | "predicate" ) identifier ":" decls
"=" expr

solve = "solve" ( "satisfy" | "minimize" | "maximize" ) expr
```

```
output = "output" ( ":" | "," ) identifier { "," identifier }
";"

decls = identifier { "," identifier } ":" type

type = ( "int" | "float" | "bool" | "set" | "array" | "tuple" )
[ "of" type ]

expr = ( "let" decls "in" expr
       | "if" expr "then" expr "else" expr
       | "case" expr "of" { ( expr "->" expr ) } [ "esac" ]
       | "forall" "(" decls ")" "," expr
       | "exists" "(" decls ")" "," expr
       | "bool_literal" | "int_literal"
       | "float_literal" | "string_literal"
       | "set_literal" | "array_literal" | "tuple_literal"
       | "identifier" | "(" expr ")"
       | expr ( "." | "->" ) identifier
       | expr ( "+" | "-" | "*" | "/" | "^" | "%" | "++" ) expr
       | expr ( "==" | "!=" | "<" | ">" | "<=" | ">=" ) expr
       | expr ( "&&" | "||" | "=>" | "xor" | "++" ) expr )
```

Además de ser un lenguaje de modelado, MiniZinc puede ser integrado con una amplia variedad de solvers. Esto incluye la integración con solvers comerciales como Gurobi y CPLEX, así como con solvers de código abierto como Chuffed y OR-Tools a través de su toolkit.

4.4.4.2 - XCSP3

XCSP3 es un formato diseñado para representar instancias de problemas restringidos de forma combinatoria, desde la perspectiva de la programación por restricciones. Está

basado en XML, un estándar para la codificación de información en formato estructurado. La utilización de este formato permite la fácil manipulación de las instancias de problemas descritas en XCSP3 y la utilización de una amplia gama de herramientas diseñadas para trabajar con este formato.

Es importante señalar que XCSP3 no es un lenguaje de modelado matemático en sí mismo, sino que es un lenguaje intermedio diseñado como representación común de problemas de satisfacción de restricciones que pueden ser utilizados por diversas herramientas y solvers.

Para facilitar el modelado, XCSP3 cuenta con librerías en varios lenguajes de programación que permiten expresar problemas de modelado y luego traducirlos a formato XML. Una de las librerías más populares es PyCSP3, que proporciona funciones para analizar y serializar archivos en formato XCSP3, así como para trabajar con funciones que describen la estructura de problemas de tipo CSP. Por lo general, los programas XCSP3 no se escriben directamente en este formato, sino que se escriben utilizando estas librerías en lenguajes de alto nivel como Java o Python.

En un programa XCSP3, un problema de satisfacción de restricciones se especifica al definir las variables, los dominios y las restricciones que lo componen. Las variables son los objetos que requieren una asignación de valor por parte del solver. El dominio es el conjunto de valores que pueden ser asignados a una de estas variables. Las restricciones, por su parte, especifican las relaciones entre las variables y limitan los dominios previamente definidos. **[33]** El siguiente código describe la gramática de XCSP3 utilizando EBNF y XML.

```
<instance format="XCSP3" type="frameworkType">
    <variables>
        (     <var.../>
          |   <array.../>
        )+
    </variables>
    <constraints>
        (    <constraints.../>
          |  <metaConstraint.../>
          |  <group.../>
          |  <block.../>
       )*
    </constraints>
    [<objectives [combination="combinationType"]>
         (    <minimize.../>
           |  <maximize.../>
         )+
     </objectives>]
</instance>
```

XCSP3 es un formato ampliamente soportado por diversos solvers de programación por restricciones y optimización. Algunos de los solvers que aceptan XCSP3 son Gecode, JaCoP, Choco, Opturion, Minion, entre otros. El formato XCSP3 proporciona una manera estandarizada de representar problemas de satisfacción de restricciones, lo que facilita la integración con una amplia variedad de solvers.

4.4.5 - INTERFAZ DE SOLVER

Al igual que con los lenguajes de modelado, hemos desarrollado una interfaz para los solucionadores. Esta interfaz facilita la expansión futura del proyecto al incorporar nuevos solucionadores de manera sencilla. Además, la interfaz es de gran importancia, ya que

nos permite aprovechar el hecho de que varios solucionadores aceptan el mismo lenguaje de modelado como entrada, lo que facilita la implementación de un gran número de solucionadores de forma eficiente y sin complicaciones.

```
class Solver(ABC):
    @abstractmethod
    def single_solution(self, Model):
        pass

    @abstractmethod
    def multiple_solutions(self, Model, num_solutions=1):
        pass

    @abstractmethod
    def count_solutions(self, Model):
        pass
```

El código anterior define una clase abstracta llamada Solver que representa un solucionador genérico. La clase contiene tres métodos abstractos: *single_solution* para obtener una única solución, *multiple_solutions* para obtener múltiples soluciones y *count_solutions* para contar el número total de soluciones. Estos métodos deben ser implementados por las subclases concretas que heredan de Solver.

Con solo implementar un solucionador que tenga MiniZinc como modelo de entrada tenemos ocho de los solver populares en la literatura: CHOCO, JaCoP, Minsat, OPL-IBM, Mistral, Gecode, Eclipse y Gurobi. Añadiendo los solvers que soporta XCSP3 se habilitan: SAT4J, PicoSAT y JavaBDD.

La implementación de estos solver es fundamental debido a la diversidad de enfoques que ofrecen, ya que cada uno utiliza diferentes algoritmos y técnicas para resolver

problemas. Esto proporciona flexibilidad a los usuarios y desarrolladores para seleccionar el solver más adecuado según sus necesidades, aumenta la robustez del sistema al contar con alternativas en caso de fallos, y permite comparar y evaluar el rendimiento de cada solver en distintos problemas y situaciones.

4.4.6 - OPERACIONES DE RAZONAMIENTO

El propósito fundamental de los elementos descritos en la tesina es llevar a cabo las operaciones de razonamiento. Su función es asegurar que un modelo de líneas de productos cumpla con las características necesarias para ser considerado correcto. Es importante destacar que un modelo puede ser válido desde un punto de vista sintáctico, pero contener errores lógicos en su construcción. Por lo tanto, la existencia de estas operaciones es crucial para identificar y corregir tanto errores sintácticos como lógicos en el modelo.

Generalmente, consideramos que una operación de razonamiento es una función pura (aquella que dadas las mismas entradas, siempre devuelve el mismo resultado y no tiene efectos secundarios) que devuelve información sobre la estructura del modelo o de alguno de sus componentes. Por ejemplo, una operación podría comprobar si un modelo de características contiene características opcionales falsas. En este caso, se verifica que la característica opcional esté presente en algunos productos y excluida de otros. Si se encuentra en todos los productos o en ninguno, esto indica que la característica debe estar vinculada mediante otra relación. Este ejemplo demuestra cómo una operación de razonamiento puede proporcionar información sobre los componentes del modelo, especialmente en términos de las conexiones que existen dentro del mismo.

Uno de los principales objetivos de nuestra propuesta de implementación es ampliar las posibilidades para los usuarios que interactúan con el sistema. Por ello, **hemos implementado dos interfaces para crear funciones de razonamiento**. El primer enfoque permite implementar estas operaciones **a través de una función pura**, que

recibe un modelo y un solucionador, y devuelve un diccionario con información sobre el modelo. La segunda opción es **mediante un lenguaje específico de dominio** (DSL) interno que diseñamos, inspirado en MapReduce, permitiendo implementar las operaciones de razonamiento de manera más intuitiva y declarativa.

Las operaciones realizadas mediante funciones puras son validadas utilizando el sistema de tipos de Python, con el objetivo de garantizar la integridad de la función. Es fundamental respetar la firma de la función, ya que esto posibilita el registro de funciones de razonamiento en el sistema de manera adecuada.

```
def validation(model: CSP, solver: Solver) -> Dict[str, Any]:
    # Function code goes here
    pass
```

En general esta API de funciones requiere que todas las funciones de razonamiento cumplan con la firma que especifica el código anterior.

La segunda API que implementa el sistema es la de MapReduce. MapReduce es un modelo de programación y un marco de procesamiento de datos basado en la abstracción de dos operaciones matemáticas fundamentales: mapeo (map) y reducción (reduce). Con esto pasaremos a dar una definición más formal de lo que hace cada una de estas funciones.

Dado un conjunto de datos de entrada, la fase de mapeo aplica una función de mapeo f: A -> B x C a cada elemento del conjunto. La función de mapeo toma un elemento de tipo A y genera un par clave-valor de tipos B y C, respectivamente. El resultado es una lista de pares clave-valor intermedios. **[34]**

Después de la fase de mapeo, se realiza una fase de "mezcla y ordenación" en la que los pares clave-valor intermedios se agrupan por clave y se ordenan según la clave. Cada

grupo de valores asociados a una clave se denomina lista de valores.

La fase de reducción toma la lista de valores agrupados y aplica una función de reducción g: B x List(C) -> D. La función de reducción procesa cada lista de valores asociados a una clave y genera un resultado de tipo D. El conjunto de resultados constituye la salida final del proceso MapReduce. **[34]**

La imagen a continuación **(Figura 3)** muestra el uso del paradigma MapReduce para contar la cantidad de letras en un conjunto de datos, ilustrando las fases de división, mapeo, ordenamiento y reducción.

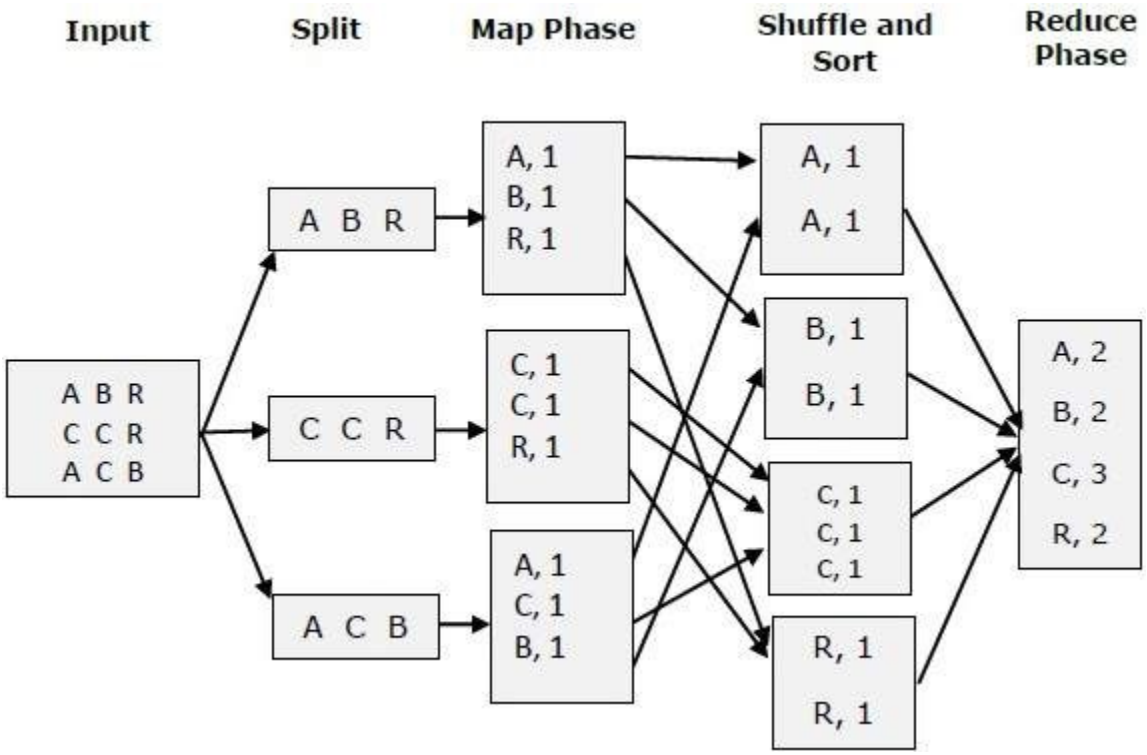

FIGURA 4.4: *Ejemplo de cómo se hacen operaciones de MapReduce.*

Nuestra implementación en el sistema es básica, pero efectiva. Aunque no explota la concurrencia que el paradigma MapReduce promete, demuestra que algunas funciones

de razonamiento populares en la literatura pueden adaptarse a este enfoque. La implementación actual del sistema incluye los métodos map y filter.

```
class MapReduce:
    def __init__(self):
        self.mapper = None
        self.filter_func = None
        self.data = []

    def map(self, func):
        self.mapper = func
        return self

    def filter(self, func):
        self.filter_func = func
        return self

    def input_data(self, data):
        self.data = data
        return self

    def execute(self):
        if not self.mapper or not self.data:
            raise ValueError(
                "Mapper, and Data must be provided when executing."
            )

        if self.filter_func:
            self.data = list(filter(self.filter_func, self.data))

        mapped_data = [self.mapper(item) for item in self.data]

        key_value_pairs = {}
        for key, value in mapped_data:
            key_value_pairs.setdefault(key, []).append(value)

        return key_value_pairs
```

El código anterior implementación básica del patrón MapReduce en Python. Permite a los usuarios definir funciones de mapeo y filtrado personalizadas para procesar datos de entrada. La clase utiliza un enfoque de "cadena fluida" para configurar y ejecutar la operación de MapReduce, lo que significa que los métodos se pueden encadenar juntos de manera legible. Durante la ejecución, la clase filtra los datos de entrada según sea necesario, mapea los datos utilizando la función de mapeo proporcionada.

Al emplear el paradigma MapReduce en ciertas funciones de razonamiento, observamos un aspecto particular: no se requiere un reducer. Esto se debe a que las operaciones de razonamiento convencionales en líneas de productos de software recorren el grafo visitando cada elemento una sola vez, lo que impide generar dos claves iguales durante la fase de mapeo. Asimismo, la operación de sort se vuelve innecesaria, ya que no es necesario agrupar datos para reducirlos.

Una operación de razonamiento clásica de los modelos de características es la de validar si existen elementos muertos en el modelo. Para ilustrar cómo funciona la API de map reduce implementaremos esta operación de razonamiento.

```
def mapper(node):
    new_model = model.add_constraint(f"Equal({node.id}, 1)")

    if not solver.single_solution(model):
        return [False, node]

    return [True, node]


map_reduce = (
    MapReduce()
        .input_data(model.nodes)
        .map(mapper)
        .filter(lambda data: data[0] == False)
)
```

```
map_reduce.execute()
```

Este código define una función llamada mapper que acepta un objeto node como argumento y crea una nueva restricción en el modelo, denominada ***new_model***. Esta restricción exige que la característica con el ID del nodo esté presente en los productos derivados. A continuación, se verifica si el modelo posee una única solución empleando la función ***single_solution*** perteneciente al objeto solver. En función del resultado, la función mapper retorna una lista que contiene un valor booleano y el objeto node. Posteriormente, se crea un objeto ***map_reduce*** utilizando la clase MapReduce, el cual toma ***model.nodes*** como datos de entrada. Este objeto aplica la función mapper a cada elemento y filtra aquellos resultados cuyo valor booleano es igual a 0. Finalmente, la operación de mapreduce se lleva a cabo mediante la ejecución de ***map_reduce.execute()***.

Otra operación de razonamiento clásica de los modelos de características es la de validar si existen falsos elementos opcionales.

```
def mapper(connection):
    new_model = model.add_constraint(
        f"Equal({connection.destination.id}, 0)"
    )

    if not solver.single_solution(model):
        return [True, node]

    return [False, node]


map_reduce = (
    MapReduce()
        .input_data(model.connections)
```

```
        .filter(lambda conn: conn.name != "optional")
        .map(mapper)
        .filter(lambda data: data[0] == False)
)

map_reduce.execute()
```

En general, nuestro objetivo es demostrar que es factible modelar las operaciones de razonamiento de líneas de productos utilizando el paradigma de MapReduce. Aunque nuestra implementación actual no aprovecha la computación distribuida, evidencia que existe el potencial de emplear este paradigma para procesar modelos de PLE de gran envergadura con la ayuda de la computación distribuida. En secciones subsiguientes, discutiremos cómo se puede mejorar esta implementación para obtener dichos beneficios **(Sección 6.1)**.

En última instancia, estas dos alternativas posibilitan la realización de todo tipo de razonamiento, como las encontradas en la literatura de líneas de productos. La aproximación implementada no solo es flexible, sino también innovadora debido a la incorporación del paradigma MapReduce en el conjunto de opciones que ofrece la plataforma, en contraste con otras soluciones analizadas en la literatura.

4.5 - APLICACIÓN EN CASO DE PRUEBA

En esta sección, analizaremos la implementación del caso descrito en el Capítulo 3 utilizando la herramienta propuesta. En primer lugar, presentaremos un modelo de características que mostrará cómo se transforma a CSP y las operaciones de razonamiento que lo acompañan. Posteriormente, ilustraremos el funcionamiento de las operaciones de razonamiento en multigrafos a través de un diagrama de transiciones.

4.5.1 - APLICACIÓN EN MODELOS DE CARACTERÍSTICAS

En esta sección, llevaremos a cabo el proceso de análisis de un modelo de características para una línea de productos de teléfonos móviles. Para ello, seguiremos los siguientes pasos: primero, modelamos el modelo de características en VariaMos y descargamos el JSON correspondiente; a continuación, crearemos un JSON que contenga las reglas de transformación que nos permitan convertir el multigrafo en un CSP; y finalmente, invocamos al sistema de solucionadores y solicitaremos la ejecución de una operación de razonamiento para el modelo específico, lo cual nos proporcionará una respuesta.

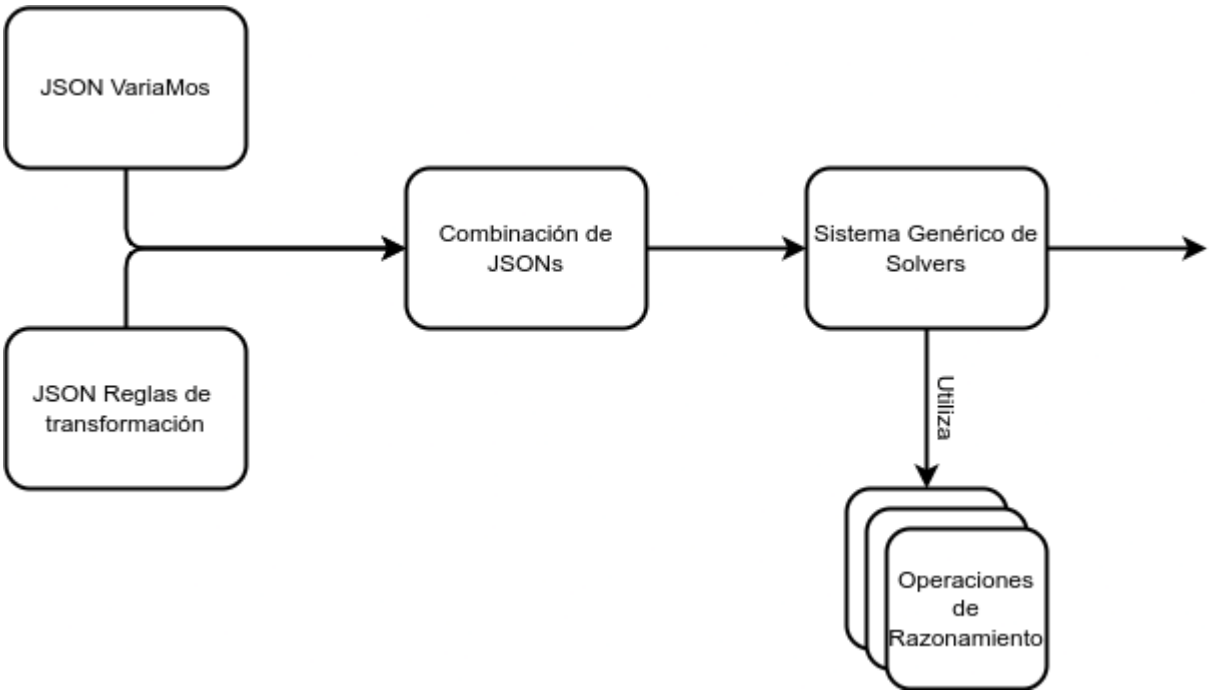

FIGURA 4.5: *Diagrama del proceso para caso de FM.*

El primer paso para poder validar el modelo es moldearlo en VariaMos. Para esto utilizaremos el modelo predeterminado llamado "Features Model" creando un nuevo tablero en blanco en el cual introduciremos nuestro modelo.

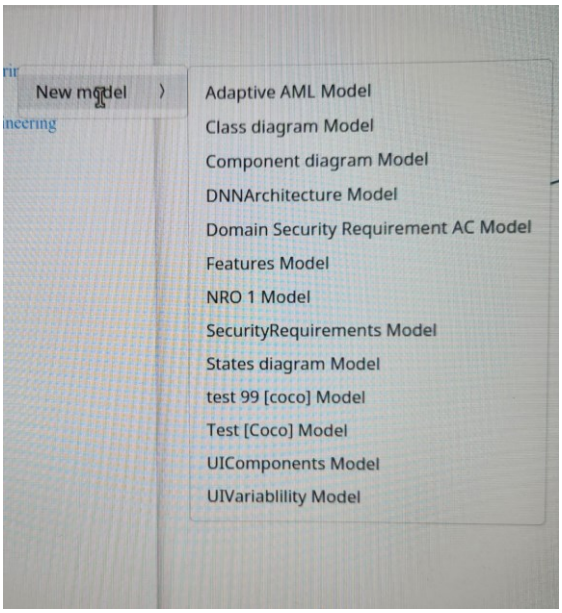

FIGURA 4.6: *Menú de opciones de modelos de dominio de VariaMos*

Luego de tener nuestro modelo listo descargamos el modelo JSON generado por la herramienta (Anexo 1 - Debido a longitud). El propósito con este JSON es poder mezclarlo con las reglas de transformación que están definidas en el siguiente JSON, esto debido a que el sistema de solvers requiere tanto el JSON generado por VariaMos

```
{
  "mapping": {
    "nodes": [
      {
        "name": "root",
        "variables": ["Integer(Node.id)"],
        "constraints": ["Equal(Node.id.value, 1)"]
      },
      {
        "name": "abstract",
        "variables": ["Integer(Node.id)"]
      },
      {
        "name": "concrete",
```

```
      "variables": ["Integer(Node.id)"]
    }
  ],
  "connections": [
    {
      "name": "mandatory",
      "constraints": [
        "Equal(
          Connection.source.value,
          Connection.destination.value
        )"
      ]
    },
    {
      "name": "optional",
      "constraints": [
        "Superior(
          Connection.source.value,
          Connection.destination.value
        )"
      ]
    },
    {
      "name": "excludes",
      "constraints": [
        "Negation(
          And(
            Equal(Connection.source.value, 1),
            Equal(Equal(Connection.destination.value, 1)
          )
        )"
      ]
    },
    {
      "name": "includes",
      "constraints": [
        "Implication(
          Equal(Connection.source.value, 1),
          Equal(Equal(Connection.destination.value, 1
```

```
        )"
      ]
    },
    {
      "name": "range",
      "constraints": [
        "Implication(
          Equal(Connection.source.value, 1),
          And(
            Inferior(
              Multiplication(
                Connection.source.value,
                Connection.min
              ),
              Addition(Connection.destination.value)
            ),
            Inferior(
              Addition(Connection.destination.value),
              Multiplication(
                Connection.source.value,
                Connection.max
              )
            )
          )"¨
      ]
    },
    {
      "name": "and",
      "constraints": [
        "Implication(
          Equal(Connection.source.value, 1),
          Equal(
            Connection.source.length,
            Addition(Connection.destination.value)
          )
        )"
      ]
    },
    {
```

```
        "name": "or",
        "constraints": [
          "Implication(
            Equal(Connection.source.value, 1),
            Superior(
              Connection.destination.length,
              Addition(Connection.destination.value)
            )
          )"
        ]
      },
      {
        "name": "xor",
        "constraints": [
          "Implication(
            Equal(Connection.source.value, 1),
            Equal(
              Connection.source.value,
              Addition(Connection.destination.value)
            )
          )"
        ]
      },
    ]
  }
}
```

El JSON proporcionado describe la estructura de reglas de transformación **(Sección 4.4.3.1)** que contiene dos listas de entidades: "nodes" (nodos) y "connections" (conexiones). Estas entidades se definen utilizando la gramática del DSL previamente proporcionada **(Sección 4.4.3)**, con restricciones y variables específicas para cada tipo de entidad.

En la sección de nodos, hay tres entidades: "root", "abstract" y "concrete". El nodo "root" tiene una variable entera llamada "Node.id" y una restricción que indica que "Node.id.value" debe ser igual a 1. Los nodos "abstract" y "concrete" solo tienen una variable entera llamada "Node.id", sin restricciones adicionales.

La sección de conexiones incluye ocho tipos diferentes de conexiones: "mandatory", "optional", "excludes", "includes", "range", "and", "or" y "xor". Cada tipo de conexión tiene restricciones específicas asociadas con ellas, basadas en la gramática del DSL.

1. La conexión ***mandatory*** requiere que el nodo padre sea igual al noda hijo
2. La conexión ***optional*** establece que el nodo padre debe ser mayor al nodo hijo
3. La conexión ***excludes*** indica que la conexión no es válida si ambas características son iguales a uno.
4. La conexión ***includes*** especifica que si alguna de las dos características pertenece al producto, entonces ambas deben estar incluidas en el.
5. La conexión ***range*** define que si la característica padre está presente, entonces el número de características hijas seleccionadas debe estar entre el valor mínimo y máximo descrito por el rango.
6. La relación ***and*** establece que si la característica padre está presente en el producto, entonces la suma de los valores de las características hijas debe ser igual al número total de características secundarias. En otras palabras, todas las características hijas deben incluirse en el producto.
7. La conexión ***or*** señala que si la característica padre es parte del producto, entonces cualquier cantidad de características hijas pueden ser seleccionadas en el mismo producto.
8. La conexión ***xor*** indica que si la característica padre existe en el producto, entonces solo una característica entre todas las hijas puede ser parte del producto.

El JSON proporciona una descripción estructurada de nodos y conexiones con restricciones y variables específicas que siguen la gramática del DSL **(Sección 4.4.3)**.

Estas restricciones y variables ayudan a definir las relaciones y propiedades de los nodos y conexiones en la estructura, permitiendo transformar el multigrafo a un CSP.

El sistema de solvers requiere que los archivos JSON de VariaMos y las reglas de transformación se envíen de manera conjunta. Para lograr esto, se crea un nuevo JSON que contiene dos claves: la primera es ***variamos_model*** y la segunda, ***mapping_rules***. A continuación, se presenta un fragmento de código que ilustra este proceso.

```
{
    "Variamos_model": {{Modelo dado por VariaMos}}
    "Mapping_rules": {{Reglas de transformación}}
}
```

Luego de tener nuestro modelo descargado de VariaMos y las reglas de transformación de estructura de grafo a CSP podemos definir las operaciones de razonamiento que nos interesan para el caso de los modelos de características. En este caso definiremos tres operaciones de razonamiento con la API de funciones puras. Las operaciones son las siguientes: "modelo vacío", "verdadero modelo de líneas de producto" y "elementos muertos".

```
def not_void_model(model: CSP, solver: Solver) -> Dict[str, Any]:
    return {
        "response": True if solver.single_solution(model) else False
    }
```

En esta función, partimos del supuesto de que si el modelo presenta al menos una solución, entonces no se considera vacío. Para llevar a cabo esta operación de razonamiento, la ejecutamos mediante una solicitud HTTP, tal como se ilustra en el fragmento de código siguiente.

```
POST /api/v1_0/csp/validation/not_void_model
```

```
{
    "response": true
}
```

Durante esta razonamiento, nos aseguramos de que disponemos de una línea de productos auténtica. Para ello, requerimos que el solver proporcione al menos dos soluciones. De este modo, garantizamos que existe variabilidad en la línea de productos.

```
def real_product_line(model: CSP, solver: Solver) -> Dict[str, Any]:
    return {
        "response": True if len(
            solver.multiple_solutions(model, num_solutions=2)
        ) == 2 else False
    }
```

De manera análoga a la razonamiento previa, podemos llevar a cabo la verificación mediante una solicitud HTTP, como se muestra a continuación.

```
POST /api/v1_0/csp/validation/real_product_line

{
    "response": true
}
```

Para confirmar la ausencia de elementos muertos, iteramos por todos los nodos del grafo. En cada nodo, lo forzamos a ser verdadero, lo que implica que si el solver proporciona al menos una solución, el elemento estará presente en al menos un producto. Sin embargo, si no se obtienen soluciones, se trata de un elemento muerto que podría ser excluido del modelo.

```
def dead_elements(model: CSP, solver: Solver) -> Dict[str, Any]:
    elements = []
```

```
    for node in model.nodes:
        new_model = model.add_constraint(f"Equal({node.id}, 1)")

        if not solver.single_solution(model):
           elements.append(node.id)

     return {
             "response": elements
     }
```

Como en las otras operaciones, podemos ejecutar esta nueva por medio de la misma estructura de request HTTP.

```
POST /api/v1_0/csp/validation/dead_elements

{
    "response": []
}
```

Para mostrar cómo se puede emplear la API de MapReduce ofrecida por el proyecto, vamos a reimplementar la operación razonamiento de elementos muertos utilizando este enfoque.

```
def mapper(node):
    new_model = model.add_constraint(f"Equal({node.id}, 1)")

    if not solver.single_solution(model):
        return [False, node]

    return [True, node]


map_reduce = (
```

```
    MapReduce()
        .input_data(model.nodes)
        .map(mapper)
        .filter(lambda data: data[0] == 0)
)

map_reduce.execute()
```

4.5.1 - APLICACIÓN EN MODELO DE TRANSICIONES

De manera análoga a cómo demostramos la posibilidad de utilizar la herramienta genérica de solvers en modelos de dominio, resulta conveniente también ilustrar cómo esta herramienta puede ser útil en escenarios de razonamiento sobre modelos de aplicación.

Para empezar, al igual que con el modelo de características, crearemos un modelo de transiciones. Una vez creado, procederemos a descargar el JSON generado por VariaMos para dicho modelo (Anexo 2 - debido a su longitud). La principal diferencia en este caso es que no es necesario establecer reglas de transformación, ya que utilizaremos únicamente la API de razonamiento estructural del multigrafo. Las operaciones de razonamiento en los modelos de aplicación no presentan variabilidad, lo que implica que, en ocasiones, las operaciones más adecuadas para el modelo no provienen de un CSP.

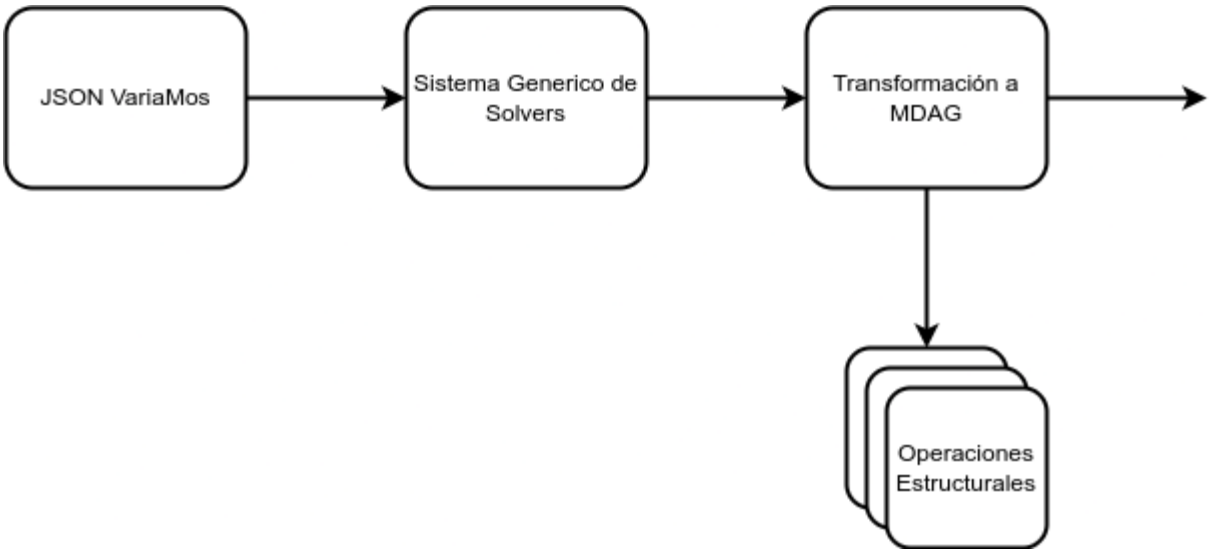

FIGURA 4.7: *Diagrama del proceso para caso de FM.*

De forma análoga al ejemplo anterior, el proceso comienza descargando el archivo JSON de VariaMos, que contiene el formato semi-estructurado para el diagrama de transiciones de un teléfono móvil. Posteriormente, este JSON se procesa en el sistema genérico de solucionadores, donde se transforma en una instancia de la clase MGraph **(Sección 4.4.2)**. Con esta instancia, procedemos a ejecutar las operaciones de razonamiento sobre el modelo para obtener los resultados correspondientes.

Una operación valiosa en este modelo particular es la de saber si el grafo es un grafo conexo. Esto es porque si en un diagrama de transiciones el grafo que lo conforma es disconexo, quiere decir que hay estados a los que es imposible acceder. Esto es un error en el modelo, ya que puede ser el caso que el autor del modelo pretendía era que existiera una comunicación para unir el grafo, o en su defecto esta era la intención del autor, por lo que puede borrar una subsección del grafo. El siguiente código muestra el procedimiento para hacer esta operación de razonamiento.

```
def is_connected(graph: MGraph) -> bool:
    visited = set()
    start_node = graph.nodes[0]

    queue = deque([start_node])

    while queue:
        current_node = queue.popleft()
        visited.add(current_node)

        connections = (
            graph.get_connections_from_source(current_node.id)
        )

        if connections:
            for connection in connections:
                for destination in connection.destinations:
                    if destination not in visited:
                        queue.append(destination)

    return len(visited) == len(graph.nodes)
```

Podemos ejecutar esta nueva por medio de la siguiente estructura de request HTTP.

```
POST /api/v1_0/csp/validation/dead_elements

{
    "response": true
}
```

De este modo, confirmamos que el modelo es, en efecto, conexo, lo que nos permite afirmar que es un modelo adecuado desde esta perspectiva.

CAPÍTULO 5

RESULTADOS

En este capítulo, se expondrán los resultados experimentales derivados de la caracterización de cada uno de los elementos del sistema, así como del sistema en su conjunto. Esta sección detalla los atributos de calidad establecidos en el capítulo previo, evaluando la propuesta en función de las métricas seleccionadas para cada atributo.

5.1 - ADECUACIÓN FUNCIONAL

Con el objetivo de medir la adecuación funcional del sistema, emplearemos tres métricas: Cobertura Funcional, Correctitud Funcional y Cobertura de Solvers Ajustada por Popularidad. Esta evaluación nos permitirá comprender en qué medida el sistema cumple con los requisitos expresados por los usuarios de las herramientas de Líneas de Productos.

5.1.1 - COBERTURA FUNCIONAL

Para medir la cobertura funcional del sistema, se llevará a cabo una comparación sistemática entre los requisitos funcionales establecidos por los diferentes actores involucrados y las funcionalidades efectivamente proporcionadas por el software, utilizando una tabla comparativa. A partir de esta tabla, se calculará el valor final de la métrica según el método descrito anteriormente.

HISTORIA DE USUARIO	CUMPLE	PARCIAL	NO CUMPLE
RF-1			
RF-2			
RF-3			
RF-4			
RF-5			
RF-6			
RF-7			
RF-8			
RF-9			
RF-10			
RF-11			
RF-12			

TABLA 5.1: *Tabla para la evaluación de camplimiento de requisitos funcionales*

Asumiremos que los requisitos no cumplidos no contribuyen a la métrica, mientras que aquellos cumplidos de manera parcial aportarán la mitad de un punto y los totalmente cumplidos recibirán un punto completo. Obtenemos entonces que esta solución tiene un **84% de cobertura funcional**.

5.1.1 - CORRECTITUD FUNCIONAL

El cálculo de la correctitud funcional se realizará de manera especial, ya que podemos obtener dos valores distintos. El primer enfoque asume que solo las funcionalidades incluidas en el proyecto se consideran para la métrica, excluyendo el requisito funcional RF-12 que sabemos que no fue implementada. Por otro lado, el segundo enfoque considera que cualquier funcionalidad requerida pero no implementada se asume como un funcionamiento incorrecto.

Los valores que determinan si las funciones devuelven el resultado esperado se obtendrán a partir del conjunto de pruebas incluido en el proyecto. El proyecto tiene un 88% de cobertura y pasa un 100% de las pruebas.

FIGURA 5.1: *Salida de herramienta de testing con valores de cobertura.*

Una vez obtenidos estos valores, es sencillo calcular cada una de las métricas correspondientes a la correctitud funcional. El primer valor de **correctitud funcional es del 100%**, mientras que el segundo **alcanza el 75.7%**.

5.1.3 - COBERTURA DE SOLVERS AJUSTADA POR POPULARIDAD

Esta métrica es la más importante que vamos a evaluar en nuestro sistema. En general, con esta métrica buscamos comprender la versatilidad de la herramienta, tanto en términos de los solvers que ofrece como en comparación con otras herramientas disponibles. Para esto vamos a hacer una tabla comparativa con otras herramientas de PLE.

HERRAMIENTA	COBERTURA

Fama	48%
FeatureIDE	17%
ProveLines	0%
Pure::Variants	5%
SPLOT	25%
VariaMos	20%
MiniZinc (Propuesta con solo)	40%
XCSP3 (Propuesta con solo)	23%
Herramienta Genérica de Solvers	**63%**

TABLA 5.2: *Tabla comparativa de completitud de solvers ajustada por popularidad para diferentes herramientas de PLE*

En general, descubrimos que la herramienta propuesta presenta una cobertura de solvers superior a la de todas las demás herramientas analizadas. Con un **63% de cobertura**, supera a FAMA, que es la herramienta más cercana en esta métrica, por un margen del 15%.

5.2 - EFICIENCIA DE DESEMPEÑO

Es inevitable que al añadir flexibilidad a un programa de software, alguna otra característica se vea afectada a cambio de estas mejoras. En el caso de esta propuesta, resulta evidente que el atributo de calidad que se ve perjudicado es la eficiencia en desempeño, ya que el software debe realizar múltiples operaciones adicionales para brindar esta flexibilidad. Nuestra intención es medir hasta qué punto se deteriora la eficiencia del sistema debido a este fenómeno.

5.2.1 - TIEMPO MEDIO DE RESPUESTA

Queriendo tener la forma más precisa posible de medir esta métrica, hemos implementado un script que genera modelos aleatorios. Con este script podremos encontrar los tiempos medios de procesamiento para cada uno de los tamaños del modelo. La idea es comparar los tiempos de evaluar el modelo solo con los solvers y sin ninguna de las capas de traducción del sistema, contra el comportamiento del sistema con todos sus módulos activos. De esta forma podemos cuantificar el sacrificio de eficiencia hecho por la generalidad de la herramienta.

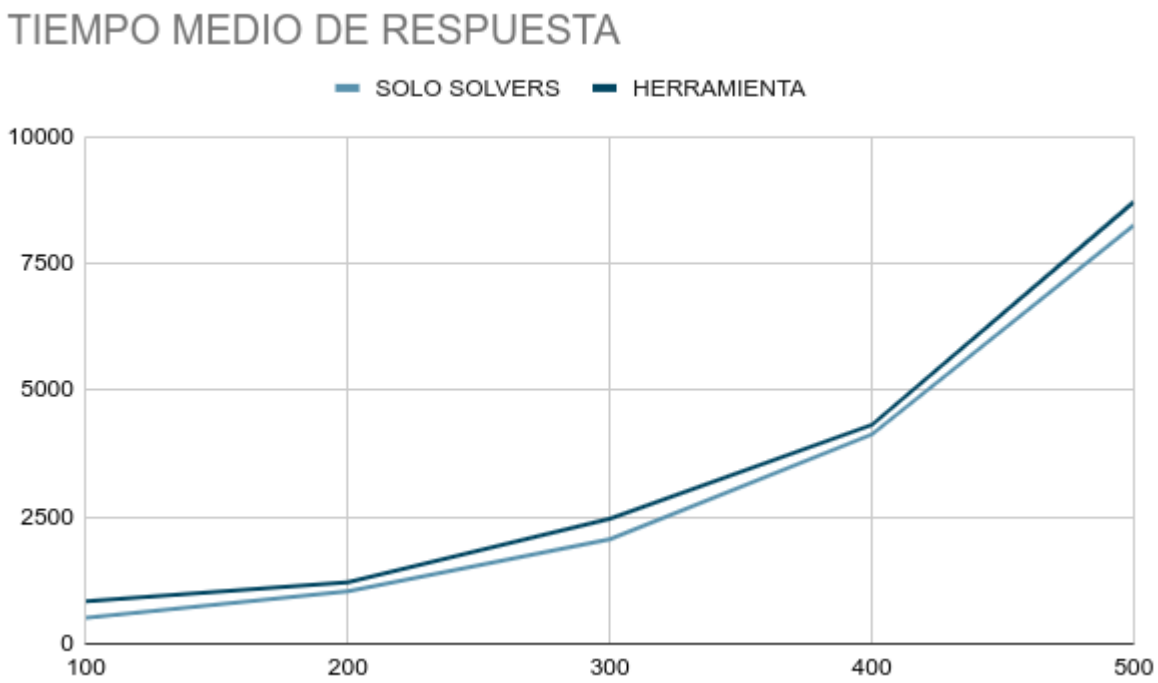

FIGURA 5.2: *Tiempo medio de respuesta para diferente tamaños de modelo*

Como se puede observar en la Figura 2, es evidente que existe una degradación en los tiempos de respuesta de la herramienta a comparación de pasar el problema

directamente al solver. Aunque la degradación es notable, resulta relativamente pequeña en comparación con el tiempo total requerido por el sistema para resolver los CSP. La intuición detrás de esto es que la degradación debería incrementarse linealmente debido al tipo de algoritmos implementados, mientras que los solvers tienen un crecimiento combinatorio en tiempo. Por lo tanto, cuanto más grande sea el modelo, menos apreciable será la degradación.

5.3 - FIABILIDAD

El propósito de medir la fiabilidad en este caso es demostrar que la herramienta fue desarrollada teniendo en cuenta las mejores prácticas, de modo que la probabilidad de experimentar degradaciones durante el uso de la herramienta sea baja. Dado que todas las métricas asociadas a este atributo de calidad se calculan automáticamente, elaboramos una tabla que muestre los valores de cada una de las métricas relacionadas.

METRICA	VALOR
Cobertura de Pruebas	88%
Adecuación de Complejidad Ciclomática	100%
Conformidad a Reglas de Código	100%

TABLA 5.3: *Tabla de resultados para métricas de fiabilidad*

5.4 - ANÁLISIS DE LA VALIDEZ DE LOS RESULTADOS

Los resultados descritos anteriormente se basan en ciertos supuestos. Nuestro objetivo es aclarar estos supuestos para que, si un lector desea replicar alguna de estas mediciones, pueda ajustar las cifras y compararlas con sus propios resultados.

En cuanto a los tiempos de respuesta, los benchmarks se ejecutaron en un portátil con un procesador Intel Core i7 y 16 GB de memoria. Cada tamaño de modelo se probó 10 veces y el resultado es el promedio de estas muestras. Aunque la variación entre las muestras es pequeña, existen factores externos que pueden afectar la medición, como las aplicaciones en segundo plano.

Respecto a la cobertura de solucionadores, utilizamos los solucionadores reportados en la literatura. Aunque existen herramientas que han evolucionado con el tiempo y cuentan con nuevos solucionadores, decidimos no incluirlos en este estudio, ya que valoramos lo que se ha publicado en la literatura científica. En particular, el caso de VariaMos es de interés, pues la herramienta actual incorpora varios de los conceptos explicados en este documento, debido a nuestra estrecha colaboración con el equipo de VariaMos.

CAPÍTULO 6

CONCLUSIONES

La implementación de un sistema genérico de solucionadores implicó la convergencia de diversos campos del conocimiento, como la programación, ingeniería de software y matemática aplicada.

El modelo que empleamos para evaluar el sistema demuestra que la solución propuesta amplía significativamente las opciones de solucionadores disponibles para los investigadores en líneas de productos de software. A pesar de que estos resultados no fueron corroborados mediante el uso continuo de la herramienta por parte de un actor industrial, los hallazgos respaldan que esta no solo es una alternativa viable, sino que también aborda de manera sustancial algunos de los problemas comúnmente encontrados en las herramientas de PLE.

En general, también se observó que la herramienta no genera una pérdida significativa en tiempos de respuesta, lo cual es importante, ya que tiempos adicionales podrían haber implicado una degradación sustancial en la experiencia del usuario.

El proceso de diseño en ingeniería permitió abordar el sistema de manera holística y encontrar una solución al problema planteado, optimizando la disponibilidad de solucionadores y minimizando los efectos no deseados de esta implementación.

En términos generales, se demostró que es viable diseñar un backend de solucionadores que pueda responder en tiempos razonables sin sacrificar la usabilidad y rendimiento del sistema.

6.1 - TRABAJO FUTURO

Aunque se obtuvieron resultados satisfactorios del sistema, todavía hay algunos aspectos que podrían mejorarse. Un caso específico es la discrepancia entre los lenguajes de representación matemática y las operaciones de validación. Mientras que la representación matemática se facilita mediante un DSL externo, las operaciones de validación se realizan en Python. Esta situación obliga a los usuarios de la herramienta a aprender dos lenguajes diferentes. Una alternativa a esto sería utilizar CLIF para la representación matemática; dado que es un dialecto de LISP, las operaciones de validación podrían implementarse en LISP, haciendo que la herramienta solo requiera el conocimiento de un único lenguaje.

En otro aspecto, la implementación del DSL interno para MapReduce no aprovecha las ventajas de concurrencia que el paradigma ofrece. Dado que algunas operaciones de validación podrían beneficiarse de este enfoque, sería conveniente analizar cómo estas ventajas en concurrencia pueden contribuir a la evaluación de modelos que, anteriormente, eran demasiado grandes para ser evaluados en tiempos razonables.

Además, un paradigma de solvers fue excluido del estudio, específicamente el CLP (Constraint Logic Programming). Sería factible incorporar alguna variante de Prolog al conjunto de solvers empleados. De esta manera, la herramienta se convertiría en la primera en ofrecer a los usuarios la capacidad de modelar problemas en cualquier paradigma de solvers.

Por último, otro aspecto que requiere mayor atención es la integración entre la herramienta genérica de solucionadores y VariaMos. En el contexto de esta investigación, la integración fue prácticamente inexistente, ya que, para probar los modelos, descargamos los archivos JSON de VariaMos y luego utilizamos Postman para realizar las solicitudes a la herramienta. Esta situación dista de ser ideal y debería ser uno de los primeros aspectos a abordar en el futuro.

LISTADO DE FIGURAS

LISTADO DE TABLAS

REFERENCIAS

[1] R. Mazo, *Guía para la Adopción industrial de Líneas de Productos de software*. Medellín: Universidad Eafit, 2018.

[2] K. Pohl, Böckle Günter, and L. F. van der, *Software product line engineering: Foundations, principles, and Techniques*. Berlin: Springer, 2005.

[3] P. Clements and L. Northrop, *Software product lines: Practices and patterns*. Boston: Addison-Wesley, 2012.

[4] T. Edward, *Foundations of Constraint Satisfaction*. Essex, England: University of Essex, 1999.

[5] E. C. Freuder, "Synthesizing constraint expressions," Communications of the ACM, vol. 21, no. 11, pp. 958–966, 1978.

[9] T. J. Schaefer, "The complexity of satisfiability problems," Proceedings of the tenth annual ACM symposium on Theory of computing - STOC ’78. ACM Press, 1978. doi: 10.1145/800133.804350**.**

[10] J. Burkardt, "DIMACS CNF," CNF files, 22-May-2008. [Online]. Available: https://people.sc.fsu.edu/~jburkardt/data/cnf/cnf.html.

[11] J. Larrosa, I. Lynce, and J. Marques-Silva, "Satisfiability: Algorithms, applications and extensions, Satisfiability: Algorithms, Applications and Extensions," 2010. [Online]. https://sat.inesc-id.pt/~ines/sac10.pdf.

[12] T. Frühwirth et al., "Constraint Logic Programming," Logic Programming in Action, Springer Berlin Heidelberg, pp. 3-35, 1992. doi: 10.1007/3-540-55930-2_2.

[13] M. Triska, "Prolog manual," SWI Prolog CLP. [Online]. https://www.swi-prolog.org/man/clpfd.html.

[14] H. W. Lenstra, "Integer programming with a fixed number of variables," Mathematics of Operations Research, vol. 8, no. 4, pp. 538–548, 1983.

[15] A. Fréville, "The multidimensional 0–1 Knapsack Problem: An overview," European Journal of Operational Research, vol. 155, no. 1, pp. 1–21, 2004.

[16] Peter J. Stuckey, Kim Marriott, Guido Tack, "The MiniZinc Handbook, MiniZinc." [Online]. Available: https://www.minizinc.org/doc-2.7.0/en/index.html.

[17] J. Franco, M. Kouril, J. Schlipf, J. Ward, S. Weaver, M. Dransfield, and W. M. Vanfleet, "SBSAT: A state-based, BDD-based satisfiability solver," Theory and Applications of Satisfiability Testing, pp. 398–410, 2004.

[18] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, and A. Jimenez, "Fama framework," 2008 12th International Software Product Line Conference, 2008.

[19] C. Kastner, T. Thum, G. Saake, J. Feigenspan, T. Leich, F. Wielgorz, and S. Apel, "Featureide: A tool framework for feature-oriented software development," 2009 IEEE 31st International Conference on Software Engineering, 2009.

[20] M. Cordy, A. Classen, P. Heymans, P.-Y. Schobbens, and A. Legay, "Provelines," Proceedings of the 17th International Software Product Line Conference co-located workshops, 2013.

[21] D. Beuche, "Pure::Variants," Systems and Software Variability Management, pp. 173–182, 2013.

[22] R. Mazo, J. C. Muñoz-Fernández, L. Rincón, C. Salinesi, and G. Tamura, "Variamos," Proceedings of the 19th International Conference on Software Product Line, 2015.

[23] L. Rincon, R. Mazo, and C. Salinesi, "Applies: A framework for evaluating organization's motivation and preparation for adopting product lines," 2018 12th International Conference on Research Challenges in Information Science (RCIS), 2018.

[24] Villota, A. *Coffee : a framework supporting expressive variability modeling and flexible automated analysis*. Technology for Human Learning. Université Panthéon-Sorbonne - Paris I, 2022.

[25] R. Barták, "Theory and Practice of Constraint Propagation," 2001

[26] A. Durán, D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés, "Flame: A formal framework for the automated analysis of Software Product Lines validated by Automated

Specification Testing," Software & Systems Modeling, vol. 16, no. 4, pp. 1049–1082, 2015.

[27] Gurov, V.S. et al. (2007) "Tools for support of automata-based programming," Programming and Computer Software, 33(6), pp. 343–355.

[28] French, M. J. *Conceptual Design for Engineers*. London: The Design Council, 1985.

[29] ISO, "ISO/IEC 25000:2014 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE", International Organization for Standardization, 2014.

[30] G. Alberto Rodríguez, *Artefactos: Diseño conceptual*. Medellín: Fondo Editorial Universidad EAFIT, 2003.

[31] R. Schiffler, "Quiver representations," *CMS Books in Mathematics*, 2014.

[32] P. Stuckey, K. Marriоћ, G. Tack, "MiniZinc Handbook 2.7.2", 2020

[33] F. Boussemart, C. Lecoutre, G. Audemard, and C. Piette, "XCSP3-core: A Format for Representing Constraint Satisfaction/Optimization Problems", CoRR, vol. abs/2009.00514, 2020.

[34] J. Dean and S. Ghemawat, “MapReduce,” Communications of the ACM, vol. 51, no. 1, pp. 107–113, 2008.

ANEXOS

Anexo 1 - JSON de modelo de características descargado de VariaMos


```
{
  "id": "a4b0bfe6-0889-4760-9bd8-5c03c2e7f134",
  "name": "NA",
  "enable": true,
  "productLines": [
    {
      "id": "e008de92-c876-4a3b-92d1-f48f85166dd0",
      "name": "NA",
      "domainEngineering": {
        "models": [
          {
            "id": "f9f71c4e-3f86-498c-8552-ac69a712f861",
            "name": "Features",
            "elements": [
              {
                "id": "12a2d27b-f02e-4232-9e87-649dbcfb6359",
                "type": "RootFeature",
                "name": "Teléfono móvil",
                "x": 520,
                "y": 140,
                "width": 100,
                "height": 33,
                "parentId": null,
                "properties": [
                  {
                    "id": "c6fe23db-2c5e-42ff-9be1-47599f2541dd",
                    "name": "Name",
                    "type": "String",
                    "custom": false,
                    "display": true
                  },
                  {
                    "id": "e48c2ca0-a094-4e0f-99ec-02c62c398b6a",
                    "name": "Selected",
                    "value": "Undefined",
```

```
            "type": "String",
            "custom": false,
            "display": true,
            "comment": "type options",
            "possibleValues": "Undefined,Selected,Unselected"
          },
          {
            "id": "53ea10b8-18a1-4868-9128-9eb820ef7a63",
            "name": "Options",
            "value": "A",
            "type": "String",
            "custom": false,
            "display": true,
            "possibleValues": "A,B,C,J"
          }
        ]
      },
      {
        "id": "2cbacbfe-4626-499b-a33f-108671bad747",
        "type": "ConcreteFeature",
        "name": "Llamadas",
        "x": 220,
        "y": 240,
        "width": 100,
        "height": 33,
        "parentId": null,
        "properties": [
          {
            "id": "0fd1d383-3c78-42d5-98eb-8fb1bf8f6b2d",
            "name": "Name",
            "type": "String",
            "custom": false,
            "display": true
          },
          {
            "id": "d223d4be-7de3-47cb-99b5-a347cddfe6b7",
            "name": "Selected",
            "value": "Undefined",
            "type": "String",
            "custom": false,
            "display": true,
            "comment": "type options",
            "possibleValues": "Undefined,Selected,Unselected"
```

```
      }
    ]
  },
  {
    "id": "e24d7e28-6137-4458-9da3-d377ea612c1d",
    "type": "ConcreteFeature",
    "name": "GPS",
    "x": 420,
    "y": 240,
    "width": 100,
    "height": 33,
    "parentId": null,
    "properties": [
      {
        "id": "b9544179-42a5-438d-9278-ab1abd531b80",
        "name": "Name",
        "type": "String",
        "custom": false,
        "display": true
      },
      {
        "id": "71613c5f-e9ad-4157-af8e-82fb88ced470",
        "name": "Selected",
        "value": "Undefined",
        "type": "String",
        "custom": false,
        "display": true,
        "comment": "type options",
        "possibleValues": "Undefined,Selected,Unselected"
      }
    ]
  },
  {
    "id": "c85b9d31-c248-44ed-b032-fb6b4ecaedc4",
    "type": "AbstractFeature",
    "name": "Pantalla",
    "x": 620,
    "y": 240,
    "width": 100,
    "height": 33,
    "parentId": null,
    "properties": [
      {
```

```
            "id": "9e29f9ad-d4b9-4794-b868-12f75bb205b2",
            "name": "Name",
            "type": "String",
            "custom": false,
            "display": true
          },
          {
            "id": "daa9816d-ca28-4950-8ffe-1486f4685eeb",
            "name": "Selected",
            "value": "Undefined",
            "type": "String",
            "custom": false,
            "display": true,
            "comment": "type options",
            "possibleValues": "Undefined,Selected,Unselected"
          }
        ]
      },
      {
        "id": "d864a8b4-d543-4802-af4d-6c5833428545",
        "type": "AbstractFeature",
        "name": "Multimedia",
        "x": 820,
        "y": 240,
        "width": 100,
        "height": 33,
        "parentId": null,
        "properties": [
          {
            "id": "b9d86016-5b9a-41b7-a47b-eb76901adf17",
            "name": "Name",
            "type": "String",
            "custom": false,
            "display": true
          },
          {
            "id": "e6cb45b1-fdc0-418f-a030-bf450ff8f5f0",
            "name": "Selected",
            "value": "Undefined",
            "type": "String",
            "custom": false,
            "display": true,
            "comment": "type options",
```

```
        "possibleValues": "Undefined,Selected,Unselected"
      }
    ]
  },
  {
    "id": "ffbb5e85-0dfa-4b2c-89ab-647c3d49ed50",
    "type": "ConcreteFeature",
    "name": "Alta resolución",
    "x": 620,
    "y": 480,
    "width": 100,
    "height": 33,
    "parentId": null,
    "properties": [
      {
        "id": "39704172-3af4-485d-b03d-534d5e4683c7",
        "name": "Name",
        "type": "String",
        "custom": false,
        "display": true
      },
      {
        "id": "4723c8d3-d7b2-4227-86ed-2f044111da9e",
        "name": "Selected",
        "value": "Undefined",
        "type": "String",
        "custom": false,
        "display": true,
        "comment": "type options",
        "possibleValues": "Undefined,Selected,Unselected"
      }
    ]
  },
  {
    "id": "54bb5fd7-f759-486d-9660-0a68b27e7c48",
    "type": "ConcreteFeature",
    "name": "Color",
    "x": 500,
    "y": 480,
    "width": 100,
    "height": 33,
    "parentId": null,
    "properties": [
```

```
        {
          "id": "73657262-0272-4162-942f-906e5d5cb356",
          "name": "Name",
          "type": "String",
          "custom": false,
          "display": true
        },
        {
          "id": "c728ec10-9fae-4e09-8e84-96374b504fed",
          "name": "Selected",
          "value": "Undefined",
          "type": "String",
          "custom": false,
          "display": true,
          "comment": "type options",
          "possibleValues": "Undefined,Selected,Unselected"
        }
      ]
    },
    {
      "id": "8fb0330c-784e-4bac-bc2d-fb9d040c0c77",
      "type": "ConcreteFeature",
      "name": "Básico",
      "x": 380,
      "y": 480,
      "width": 100,
      "height": 33,
      "parentId": null,
      "properties": [
        {
          "id": "af04e779-dda1-4ffb-bd85-e905e962a61e",
          "name": "Name",
          "type": "String",
          "custom": false,
          "display": true
        },
        {
          "id": "aa3f4431-1e1b-4527-a1ca-72ea17554ac2",
          "name": "Selected",
          "value": "Undefined",
          "type": "String",
          "custom": false,
          "display": true,
```

```
        "comment": "type options",
        "possibleValues": "Undefined,Selected,Unselected"
      }
    ]
  },
  {
    "id": "b2277ca2-4592-4a34-b02b-38214fb4aecb",
    "type": "ConcreteFeature",
    "name": "Cámara",
    "x": 820,
    "y": 480,
    "width": 100,
    "height": 33,
    "parentId": null,
    "properties": [
      {
        "id": "2d5d7274-6d8e-4330-ae70-fd49bfd9f2c7",
        "name": "Name",
        "type": "String",
        "custom": false,
        "display": true
      },
      {
        "id": "3983b075-0c3e-42eb-b28d-5e6220629768",
        "name": "Selected",
        "value": "Undefined",
        "type": "String",
        "custom": false,
        "display": true,
        "comment": "type options",
        "possibleValues": "Undefined,Selected,Unselected"
      }
    ]
  },
  {
    "id": "264e555c-c557-4b05-88f2-76e56469b31e",
    "type": "ConcreteFeature",
    "name": "MP3",
    "x": 940,
    "y": 480,
    "width": 100,
    "height": 33,
    "parentId": null,
```

```
      "properties": [
        {
          "id": "ed4607b4-6fb8-4afa-a86e-99c6cec06b35",
          "name": "Name",
          "type": "String",
          "custom": false,
          "display": true
        },
        {
          "id": "7ded4fad-f9e8-49c0-b7d0-e1e55d15ce32",
          "name": "Selected",
          "value": "Undefined",
          "type": "String",
          "custom": false,
          "display": true,
          "comment": "type options",
          "possibleValues": "Undefined,Selected,Unselected"
        }
      ]
    },
    {
      "id": "eedde547-0b35-4687-aa7f-d580e53e9e28",
      "type": "Bundle",
      "name": "Bundle 1",
      "x": 820,
      "y": 340,
      "width": 100,
      "height": 50,
      "parentId": null,
      "properties": [
        {
          "id": "7aff71a6-6d2e-48e0-a14b-572f390dbcdb",
          "name": "Name",
          "type": "String",
          "custom": false,
          "display": true
        },
        {
          "id": "94dc923a-9233-4cc3-83b5-403e545105a6",
          "name": "Type",
          "value": "Or",
          "type": "String",
          "custom": false,
```

```
            "display": true,
            "comment": "type options",
            "possibleValues": "And,Or,Xor,Range"
          },
          {
            "id": "e490c4ec-7429-42a6-b8be-e574b27a0c0e",
            "name": "RangeMin",
            "type": "String",
            "linked_property": "Type",
            "linked_value": "Range",
            "custom": false,
            "display": false
          },
          {
            "id": "c745df82-1fd2-4ed8-8d03-96981b71bc51",
            "name": "RangeMax",
            "type": "String",
            "linked_property": "Type",
            "linked_value": "Range",
            "custom": false,
            "display": false
          },
          {
            "id": "35b516e8-fe49-4d8c-baaa-fa141a4b35a5",
            "name": "Testing",
            "type": "Integer",
            "custom": false,
            "display": true,
            "comment": "prueba",
            "possibleValues": "1..10"
          }
        ]
      },
      {
        "id": "3a434b7d-af6c-47f9-bcf2-cb11cf1584f1",
        "type": "Bundle",
        "name": "Bundle 1",
        "x": 620,
        "y": 340,
        "width": 100,
        "height": 50,
        "parentId": null,
        "properties": [
```

```
{
  "id": "e7fedae5-b66c-4e77-9588-cc1cf6e49b67",
  "name": "Name",
  "type": "String",
  "custom": false,
  "display": true
},
{
  "id": "c75b8fbe-7557-46c1-a8cc-b35a001fb55c",
  "name": "Type",
  "value": "Xor",
  "type": "String",
  "custom": false,
  "display": true,
  "comment": "type options",
  "possibleValues": "And,Or,Xor,Range"
},
{
  "id": "2e442d4e-bf08-42c7-a9ea-1f60327ff4e7",
  "name": "RangeMin",
  "type": "String",
  "linked_property": "Type",
  "linked_value": "Range",
  "custom": false,
  "display": false
},
{
  "id": "628dd070-8a72-425c-a7e1-758b34d51907",
  "name": "RangeMax",
  "type": "String",
  "linked_property": "Type",
  "linked_value": "Range",
  "custom": false,
  "display": false
},
{
  "id": "d08c8406-b5f2-4f5d-a3ea-5a733715ca84",
  "name": "Testing",
  "type": "Integer",
  "custom": false,
  "display": true,
  "comment": "prueba",
  "possibleValues": "1..10"
```

```
        }
      ]
    }
  ],
  "relationships": [
    {
      "id": "b4f2ec14-ab9d-494c-ba0c-be8a22feaa0a",
      "type": "AbstractFeature_Bundle",
      "name": "Media_And",
      "sourceId": "d864a8b4-d543-4802-af4d-6c5833428545",
      "targetId": "eedde547-0b35-4687-aa7f-d580e53e9e28",
      "points": [],
      "min": 0,
      "max": 1,
      "properties": []
    },
    {
      "id": "c0575b47-c718-40bd-928b-b59135fec0f8",
      "type": "Bundle_Feature",
      "name": "And_ConcreteFeature 1",
      "sourceId": "eedde547-0b35-4687-aa7f-d580e53e9e28",
      "targetId": "b2277ca2-4592-4a34-b02b-38214fb4aecb",
      "points": [],
      "min": 0,
      "max": 9999999,
      "properties": []
    },
    {
      "id": "5b5dc223-c981-4891-8bbd-34be5bf36373",
      "type": "Bundle_Feature",
      "name": "And_ConcreteFeature 1",
      "sourceId": "eedde547-0b35-4687-aa7f-d580e53e9e28",
      "targetId": "264e555c-c557-4b05-88f2-76e56469b31e",
      "points": [],
      "min": 0,
      "max": 9999999,
      "properties": []
    },
    {
      "id": "87658a86-a039-42f7-b9f1-8e2fb306f16c",
      "type": "AbstractFeature_Bundle",
      "name": "Screen_And",
      "sourceId": "c85b9d31-c248-44ed-b032-fb6b4ecaedc4",
```

```
      "targetId": "3a434b7d-af6c-47f9-bcf2-cb11cf1584f1",
      "points": [],
      "min": 0,
      "max": 1,
      "properties": []
    },
    {
      "id": "8abdd7db-d66b-468e-b2aa-0a9edbce30f8",
      "type": "Bundle_Feature",
      "name": "And_ConcreteFeature 1",
      "sourceId": "3a434b7d-af6c-47f9-bcf2-cb11cf1584f1",
      "targetId": "ffbb5e85-0dfa-4b2c-89ab-647c3d49ed50",
      "points": [],
      "min": 0,
      "max": 9999999,
      "properties": []
    },
    {
      "id": "34defdaa-d2a8-4593-a203-4b0196635240",
      "type": "Bundle_Feature",
      "name": "And_ConcreteFeature 1",
      "sourceId": "3a434b7d-af6c-47f9-bcf2-cb11cf1584f1",
      "targetId": "54bb5fd7-f759-486d-9660-0a68b27e7c48",
      "points": [],
      "min": 0,
      "max": 9999999,
      "properties": []
    },
    {
      "id": "3106d89c-7330-4ada-809f-43af6c06af98",
      "type": "Bundle_Feature",
      "name": "And_ConcreteFeature 1",
      "sourceId": "3a434b7d-af6c-47f9-bcf2-cb11cf1584f1",
      "targetId": "8fb0330c-784e-4bac-bc2d-fb9d040c0c77",
      "points": [],
      "min": 0,
      "max": 9999999,
      "properties": []
    },
    {
      "id": "ada202c1-ac0e-4ab9-baf5-102f8f478649",
      "type": "ConcreteFeature_Feature",
      "name": "ConcreteFeature 1_ConcreteFeature 1",
```

```
                "sourceId": "b2277ca2-4592-4a34-b02b-38214fb4aecb",
                "targetId": "ffbb5e85-0dfa-4b2c-89ab-647c3d49ed50",
                "points": [],
                "min": 0,
                "max": 9999999,
                "properties": [
                  {
                    "id": "aec1ea89-b4c8-4a65-9388-367a1b2ed6c1",
                    "name": "Type",
                    "value": "Includes",
                    "type": "String",
                    "custom": false,
                    "display": true,
                    "possibleValues":
"Mandatory,Optional,Includes,Excludes,IndividualCardinality"
                  },
                  {
                    "id": "4ffbf6f7-cc13-492d-9407-9370a2fd3809",
                    "name": "MinValue",
                    "type": "String",
                    "linked_property": "Type",
                    "linked_value": "IndividualCardinality",
                    "custom": false,
                    "display": false
                  },
                  {
                    "id": "a66c83e4-d9ca-47ce-9f0f-85f6ffd04168",
                    "name": "MaxValue",
                    "type": "String",
                    "linked_property": "Type",
                    "linked_value": "IndividualCardinality",
                    "custom": false,
                    "display": false
                  }
                ]
              },
              {
                "id": "28fb7e6c-1f96-4878-b4c2-f9bba65d2df8",
                "type": "ConcreteFeature_Feature",
                "name": "ConcreteFeature 1_GPS",
                "sourceId": "8fb0330c-784e-4bac-bc2d-fb9d040c0c77",
                "targetId": "e24d7e28-6137-4458-9da3-d377ea612c1d",
                "points": [],
```

```
                "min": 0,
                "max": 9999999,
                "properties": [
                  {
                    "id": "4c01d45d-a902-4444-81e0-54e39624d270",
                    "name": "Type",
                    "value": "Excludes",
                    "type": "String",
                    "custom": false,
                    "display": true,
                    "possibleValues":
"Mandatory,Optional,Includes,Excludes,IndividualCardinality"
                  },
                  {
                    "id": "88b27ec3-0c9f-49aa-89e6-63c223db720a",
                    "name": "MinValue",
                    "type": "String",
                    "linked_property": "Type",
                    "linked_value": "IndividualCardinality",
                    "custom": false,
                    "display": false
                  },
                  {
                    "id": "37d4733e-d6c1-4cc1-8cb8-6599934a2596",
                    "name": "MaxValue",
                    "type": "String",
                    "linked_property": "Type",
                    "linked_value": "IndividualCardinality",
                    "custom": false,
                    "display": false
                  }
                ]
              },
              {
                "id": "b1c57297-88ca-4d23-a203-8a6a051808c4",
                "type": "RootFeature_Feature",
                "name": "Mobile Phone_Calls",
                "sourceId": "12a2d27b-f02e-4232-9e87-649dbcfb6359",
                "targetId": "2cbacbfe-4626-499b-a33f-108671bad747",
                "points": [],
                "min": 0,
                "max": 9999999,
                "properties": [
```

```
                {
                  "id": "ee45567b-7162-4ba5-8d07-83167f3d9119",
                  "name": "Type",
                  "value": "Mandatory",
                  "type": "String",
                  "custom": false,
                  "display": true,
                  "possibleValues":
"Mandatory,Optional,Includes,Excludes,IndividualCardinality"
                },
                {
                  "id": "3de5a958-83c5-4471-8708-47dbb67740b3",
                  "name": "MinValue",
                  "type": "String",
                  "linked_property": "Type",
                  "linked_value": "IndividualCardinality",
                  "custom": false,
                  "display": false
                },
                {
                  "id": "3f169f4b-3d92-411a-a3a9-be9215338898",
                  "name": "MaxValue",
                  "type": "String",
                  "linked_property": "Type",
                  "linked_value": "IndividualCardinality",
                  "custom": false,
                  "display": false
                }
              ]
            },
            {
              "id": "1700c777-96dc-4e7a-ba87-4452251797f3",
              "type": "RootFeature_Feature",
              "name": "Mobile Phone_GPS",
              "sourceId": "12a2d27b-f02e-4232-9e87-649dbcfb6359",
              "targetId": "e24d7e28-6137-4458-9da3-d377ea612c1d",
              "points": [],
              "min": 0,
              "max": 9999999,
              "properties": [
                {
                  "id": "4ee997e7-8dbc-4336-a2f4-12fe05133e0b",
                  "name": "Type",
```

```
                    "value": "Optional",
                    "type": "String",
                    "custom": false,
                    "display": true,
                    "possibleValues":
"Mandatory,Optional,Includes,Excludes,IndividualCardinality"
                  },
                  {
                    "id": "5448a817-bfcf-4da0-94c7-c2720b179019",
                    "name": "MinValue",
                    "type": "String",
                    "linked_property": "Type",
                    "linked_value": "IndividualCardinality",
                    "custom": false,
                    "display": false
                  },
                  {
                    "id": "d581bffe-1418-42f1-8d88-60763fc88e4a",
                    "name": "MaxValue",
                    "type": "String",
                    "linked_property": "Type",
                    "linked_value": "IndividualCardinality",
                    "custom": false,
                    "display": false
                  }
                ]
              },
              {
                "id": "800fd9c6-2298-4604-aef6-106563c68147",
                "type": "RootFeature_Feature",
                "name": "Mobile Phone_Screen",
                "sourceId": "12a2d27b-f02e-4232-9e87-649dbcfb6359",
                "targetId": "c85b9d31-c248-44ed-b032-fb6b4ecaedc4",
                "points": [],
                "min": 0,
                "max": 9999999,
                "properties": [
                  {
                    "id": "2f1d2085-e850-4902-8c8f-0c98e40cfe54",
                    "name": "Type",
                    "value": "Mandatory",
                    "type": "String",
                    "custom": false,
```

```
                    "display": true,
                    "possibleValues":
"Mandatory,Optional,Includes,Excludes,IndividualCardinality"
                  },
                  {
                    "id": "80397ef7-204a-4e10-ba0c-978e29df811a",
                    "name": "MinValue",
                    "type": "String",
                    "linked_property": "Type",
                    "linked_value": "IndividualCardinality",
                    "custom": false,
                    "display": false
                  },
                  {
                    "id": "3442ae9f-d773-4c30-a5a3-ae19c7cf9c90",
                    "name": "MaxValue",
                    "type": "String",
                    "linked_property": "Type",
                    "linked_value": "IndividualCardinality",
                    "custom": false,
                    "display": false
                  }
                ]
              },
              {
                "id": "04fbc801-6363-49db-9363-25061dc1b5e8",
                "type": "RootFeature_Feature",
                "name": "Mobile Phone_Media",
                "sourceId": "12a2d27b-f02e-4232-9e87-649dbcfb6359",
                "targetId": "d864a8b4-d543-4802-af4d-6c5833428545",
                "points": [],
                "min": 0,
                "max": 9999999,
                "properties": [
                  {
                    "id": "be476a54-de4c-4315-9e88-ead13fc21ce5",
                    "name": "Type",
                    "value": "Optional",
                    "type": "String",
                    "custom": false,
                    "display": true,
                    "possibleValues":
"Mandatory,Optional,Includes,Excludes,IndividualCardinality"
```

```
                },
                {
                  "id": "1c480ac4-9886-4c49-9a7d-fdd65550ef83",
                  "name": "MinValue",
                  "type": "String",
                  "linked_property": "Type",
                  "linked_value": "IndividualCardinality",
                  "custom": false,
                  "display": false
                },
                {
                  "id": "0de0c82e-7d16-41fb-b910-89c42ed7068c",
                  "name": "MaxValue",
                  "type": "String",
                  "linked_property": "Type",
                  "linked_value": "IndividualCardinality",
                  "custom": false,
                  "display": false
                }
              ]
            }
          ]
        }
      ],
      "languagesAllowed": []
    },
    "applicationEngineering": {
      "models": [],
      "languagesAllowed": [],
      "applications": []
    }
  }
]
}
```


Anexo 2 - JSON de modelo de transiciones descargado de VariaMos


```
{
  "id": "fe8ee984-38eb-4ea6-aeba-2da13c245a18",
  "name": "PL",
  "enable": true,
```

```
"productLines": [
  {
    "id": "f9f0f828-ed73-4e51-863e-c133678e329b",
    "name": "PL",
    "domainEngineering": {
      "models": [
        {
          "id": "b1dfa806-1068-4aad-8251-ef1bf73d0089",
          "name": "States diagram",
          "elements": [
            {
              "id": "5558d9e3-cfc2-40e8-be64-d08e54e217af",
              "type": "InitialState",
              "name": "inactivo",
              "x": 600,
              "y": 100,
              "width": 100,
              "height": 35,
              "parentId": null,
              "properties": [
                {
                  "id": "4e68a116-5467-41cc-a4ba-65ee1c8bc2c3",
                  "name": "Selected",
                  "value": "Undefined",
                  "type": "String",
                  "custom": false,
                  "display": true,
                  "comment": "type options",
                  "possibleValues": "Undefined,Selected,Unselected"
                }
              ]
            },
            {
              "id": "789f88f8-cb99-4d31-ab4b-16505d72ceb6",
              "type": "Transition",
              "name": "llamada",
              "x": 330,
              "y": 220,
              "width": 100,
              "height": 25,
              "parentId": null,
              "properties": [
                {
```

```
            "id": "03703bad-0f86-44ca-9113-23d5f8374700",
            "name": "Selected",
            "value": "Undefined",
            "type": "String",
            "custom": false,
            "display": true,
            "comment": "type options",
            "possibleValues": "Undefined,Selected,Unselected"
          }
        ]
      },
      {
        "id": "a26e8f91-4edf-4078-916e-8b185c6c4e54",
        "type": "Transition",
        "name": "menu",
        "x": 330,
        "y": 180,
        "width": 100,
        "height": 25,
        "parentId": null,
        "properties": [
          {
            "id": "fe4ee373-f498-48b4-b311-161fe5a31c08",
            "name": "Selected",
            "value": "Undefined",
            "type": "String",
            "custom": false,
            "display": true,
            "comment": "type options",
            "possibleValues": "Undefined,Selected,Unselected"
          }
        ]
      },
      {
        "id": "c8da9c81-b0a8-4f18-8a9e-29da0460533b",
        "type": "Transition",
        "name": "salida",
        "x": 330,
        "y": 130,
        "width": 100,
        "height": 25,
        "parentId": null,
        "properties": [
```

```
        {
          "id": "ed10a87f-1286-4d7d-aa8f-a9b5db9920f8",
          "name": "Selected",
          "value": "Undefined",
          "type": "String",
          "custom": false,
          "display": true,
          "comment": "type options",
          "possibleValues": "Undefined,Selected,Unselected"
        }
      ]
    },
    {
      "id": "f56da883-e423-4da9-85db-34f2ae536275",
      "type": "State",
      "name": "menu",
      "x": 70,
      "y": 190,
      "width": 100,
      "height": 35,
      "parentId": null,
      "properties": [
        {
          "id": "fa704834-27a1-480d-938e-9c7d44e13b12",
          "name": "Selected",
          "value": "Undefined",
          "type": "String",
          "custom": false,
          "display": true,
          "comment": "type options",
          "possibleValues": "Undefined,Selected,Unselected"
        }
      ]
    },
    {
      "id": "eacdad1e-112a-4491-a27f-6ab5ba455db7",
      "type": "Transition",
      "name": "llamada",
      "x": 330,
      "y": 260,
      "width": 100,
      "height": 25,
      "parentId": null,
```

```
      "properties": [
        {
          "id": "6c50a17b-8f37-48db-90ac-8973282b33b3",
          "name": "Selected",
          "value": "Undefined",
          "type": "String",
          "custom": false,
          "display": true,
          "comment": "type options",
          "possibleValues": "Undefined,Selected,Unselected"
        }
      ]
    },
    {
      "id": "db876abe-dfd3-446a-9723-9961f1613395",
      "type": "State",
      "name": "llamando",
      "x": 860,
      "y": 250,
      "width": 100,
      "height": 35,
      "parentId": null,
      "properties": [
        {
          "id": "e126556f-8b36-433f-8ae1-2e21d41fafba",
          "name": "Selected",
          "value": "Undefined",
          "type": "String",
          "custom": false,
          "display": true,
          "comment": "type options",
          "possibleValues": "Undefined,Selected,Unselected"
        }
      ]
    },
    {
      "id": "595c0cca-3130-45ef-8e51-78ee09e8073b",
      "type": "Transition",
      "name": "salida",
      "x": 860,
      "y": 190,
      "width": 100,
      "height": 25,
```

```
      "parentId": null,
      "properties": [
        {
          "id": "a3a7563d-dab3-4458-b919-ef323e2386fd",
          "name": "Selected",
          "value": "Undefined",
          "type": "String",
          "custom": false,
          "display": true,
          "comment": "type options",
          "possibleValues": "Undefined,Selected,Unselected"
        }
      ]
    },
    {
      "id": "a373c397-d72f-4472-8d68-dcbd7a35c1fd",
      "type": "State",
      "name": "hablando",
      "x": 340,
      "y": 430,
      "width": 100,
      "height": 35,
      "parentId": null,
      "properties": [
        {
          "id": "2b9e66ab-10a1-427d-8704-b069845c131c",
          "name": "Selected",
          "value": "Undefined",
          "type": "String",
          "custom": false,
          "display": true,
          "comment": "type options",
          "possibleValues": "Undefined,Selected,Unselected"
        }
      ]
    },
    {
      "id": "b8f1e05f-4825-45c6-9d55-04462b00ff89",
      "type": "State",
      "name": "sonando",
      "x": 860,
      "y": 420,
      "width": 100,
```

```
      "height": 35,
      "parentId": null,
      "properties": [
        {
          "id": "a065add0-cf8f-4d4c-b679-6b7b6c685f34",
          "name": "Selected",
          "value": "Undefined",
          "type": "String",
          "custom": false,
          "display": true,
          "comment": "type options",
          "possibleValues": "Undefined,Selected,Unselected"
        }
      ]
    },
    {
      "id": "44844e1c-2d6a-4201-a03c-d1d3e45ef69f",
      "type": "Transition",
      "name": "contestado",
      "x": 600,
      "y": 340,
      "width": 100,
      "height": 25,
      "parentId": null,
      "properties": [
        {
          "id": "0c7e940b-b47a-4037-a2ea-32a1ed498e08",
          "name": "Selected",
          "value": "Undefined",
          "type": "String",
          "custom": false,
          "display": true,
          "comment": "type options",
          "possibleValues": "Undefined,Selected,Unselected"
        }
      ]
    },
    {
      "id": "e262afb4-9a8e-420f-b443-1c6ba9e68dde",
      "type": "Transition",
      "name": "llamada",
      "x": 600,
      "y": 430,
```

```
      "width": 100,
      "height": 25,
      "parentId": null,
      "properties": [
        {
          "id": "40f32485-cc54-4eec-8454-78011f4e31e6",
          "name": "Selected",
          "value": "Undefined",
          "type": "String",
          "custom": false,
          "display": true,
          "comment": "type options",
          "possibleValues": "Undefined,Selected,Unselected"
        }
      ]
    },
    {
      "id": "157d73b7-8567-43ec-bb48-9f7de3a5b33a",
      "type": "Transition",
      "name": "Transition 1",
      "x": 770,
      "y": 340,
      "width": 100,
      "height": 25,
      "parentId": "4a5db719-ad25-4ddc-8dff-d1ca90ed70ad",
      "properties": [
        {
          "id": "a9b5cabc-ecb5-4d8d-b32d-f555d8301f83",
          "name": "Selected",
          "value": "Undefined",
          "type": "String",
          "custom": false,
          "display": true,
          "comment": "type options",
          "possibleValues": "Undefined,Selected,Unselected"
        }
      ]
    },
    {
      "id": "8431e33c-d9b5-4661-abbe-c97c30a99b46",
      "type": "Transition",
      "name": "salida",
      "x": 700,
```

```
      "y": 220,
      "width": 100,
      "height": 25,
      "parentId": null,
      "properties": [
        {
          "id": "2bbe6845-b548-4b85-b4ec-f16aa718c93f",
          "name": "Selected",
          "value": "Undefined",
          "type": "String",
          "custom": false,
          "display": true,
          "comment": "type options",
          "possibleValues": "Undefined,Selected,Unselected"
        }
      ]
    },
    {
      "id": "190c92a3-2cbd-4ffa-9c86-0c395ff6053e",
      "type": "Transition",
      "name": "salida",
      "x": 500,
      "y": 220,
      "width": 100,
      "height": 25,
      "parentId": null,
      "properties": [
        {
          "id": "5b387d61-32dc-4781-a52d-3781a9542db7",
          "name": "Selected",
          "value": "Undefined",
          "type": "String",
          "custom": false,
          "display": true,
          "comment": "type options",
          "possibleValues": "Undefined,Selected,Unselected"
        }
      ]
    }
  ],
  "relationships": [
    {
      "id": "87eea8d1-4908-421d-ae9a-655bcb7a5323",
```

```
      "type": "InitialState_Transition",
      "name": "InitialState 1_Transition 1",
      "sourceId": "5558d9e3-cfc2-40e8-be64-d08e54e217af",
      "targetId": "789f88f8-cb99-4d31-ab4b-16505d72ceb6",
      "points": [],
      "min": 1,
      "max": 9999999,
      "properties": []
    },
    {
      "id": "90288981-5c6d-4d0c-b91a-9db3be12ff59",
      "type": "Transition_State",
      "name": "Transition 1_State 1",
      "sourceId": "789f88f8-cb99-4d31-ab4b-16505d72ceb6",
      "targetId": "f56da883-e423-4da9-85db-34f2ae536275",
      "points": [],
      "min": 1,
      "max": 1,
      "properties": []
    },
    {
      "id": "bda8ef9e-8930-49ea-b0d8-c2a6fa43d67a",
      "type": "InitialState_Transition",
      "name": "InitialState 1_Transition 1",
      "sourceId": "5558d9e3-cfc2-40e8-be64-d08e54e217af",
      "targetId": "a26e8f91-4edf-4078-916e-8b185c6c4e54",
      "points": [],
      "min": 1,
      "max": 9999999,
      "properties": []
    },
    {
      "id": "e8998040-aa3f-466c-8aca-d8da435e8c05",
      "type": "Transition_State",
      "name": "Transition 1_State 1",
      "sourceId": "a26e8f91-4edf-4078-916e-8b185c6c4e54",
      "targetId": "f56da883-e423-4da9-85db-34f2ae536275",
      "points": [],
      "min": 1,
      "max": 1,
      "properties": []
    },
    {
```

```
      "id": "9a14d30c-5ba3-43c7-80b9-6b3bdd70a33f",
      "type": "State_Transition",
      "name": "State 1_Transition 1",
      "sourceId": "f56da883-e423-4da9-85db-34f2ae536275",
      "targetId": "c8da9c81-b0a8-4f18-8a9e-29da0460533b",
      "points": [],
      "min": 1,
      "max": 9999999,
      "properties": []
    },
    {
      "id": "cb2f1acf-b0c7-40d3-80b8-4da9f61ea244",
      "type": "Transition_State",
      "name": "Transition 1_InitialState 1",
      "sourceId": "c8da9c81-b0a8-4f18-8a9e-29da0460533b",
      "targetId": "5558d9e3-cfc2-40e8-be64-d08e54e217af",
      "points": [],
      "min": 1,
      "max": 1,
      "properties": []
    },
    {
      "id": "144187e8-f963-499c-af5d-16691a66fe66",
      "type": "State_Transition",
      "name": "menu_llamada",
      "sourceId": "f56da883-e423-4da9-85db-34f2ae536275",
      "targetId": "eacdad1e-112a-4491-a27f-6ab5ba455db7",
      "points": [],
      "min": 1,
      "max": 9999999,
      "properties": []
    },
    {
      "id": "861459c3-690f-44db-8b39-0218eb151ebb",
      "type": "Transition_State",
      "name": "llamada_llamando",
      "sourceId": "eacdad1e-112a-4491-a27f-6ab5ba455db7",
      "targetId": "db876abe-dfd3-446a-9723-9961f1613395",
      "points": [],
      "min": 1,
      "max": 1,
      "properties": []
    },
```

```
        {
          "id": "e688b5a0-fef7-4a01-8761-634a39ab1842",
          "type": "State_Transition",
          "name": "llamando_Transition 1",
          "sourceId": "db876abe-dfd3-446a-9723-9961f1613395",
          "targetId": "595c0cca-3130-45ef-8e51-78ee09e8073b",
          "points": [],
          "min": 1,
          "max": 9999999,
          "properties": []
        },
        {
          "id": "fa6f1174-4571-4120-9e87-a2008626aea7",
          "type": "Transition_State",
          "name": "Transition 1_inactivo",
          "sourceId": "595c0cca-3130-45ef-8e51-78ee09e8073b",
          "targetId": "5558d9e3-cfc2-40e8-be64-d08e54e217af",
          "points": [],
          "min": 1,
          "max": 1,
          "properties": []
        },
        {
          "id": "6b14a0ba-8dcd-406e-a4d9-79b57559e0ca",
          "type": "Transition_State",
          "name": "contestado_hablando",
          "sourceId": "44844e1c-2d6a-4201-a03c-d1d3e45ef69f",
          "targetId": "a373c397-d72f-4472-8d68-dcbd7a35c1fd",
          "points": [],
          "min": 1,
          "max": 1,
          "properties": []
        },
        {
          "id": "bf608a56-4bbb-474b-bbe4-687dcd0aacde",
          "type": "State_Transition",
          "name": "sonando_llamada",
          "sourceId": "b8f1e05f-4825-45c6-9d55-04462b00ff89",
          "targetId": "e262afb4-9a8e-420f-b443-1c6ba9e68dde",
          "points": [],
          "min": 1,
          "max": 9999999,
          "properties": []
```

```
        },
        {
          "id": "9cf3d549-3709-4e33-9958-104f15530b07",
          "type": "Transition_State",
          "name": "llamada_hablando",
          "sourceId": "e262afb4-9a8e-420f-b443-1c6ba9e68dde",
          "targetId": "a373c397-d72f-4472-8d68-dcbd7a35c1fd",
          "points": [],
          "min": 1,
          "max": 1,
          "properties": []
        },
        {
          "id": "aa98f3aa-437e-45ae-9b02-47718408ac1b",
          "type": "State_Transition",
          "name": "llamando_contestado",
          "sourceId": "db876abe-dfd3-446a-9723-9961f1613395",
          "targetId": "44844e1c-2d6a-4201-a03c-d1d3e45ef69f",
          "points": [],
          "min": 1,
          "max": 9999999,
          "properties": []
        },
        {
          "id": "b70e5e3d-e247-41bd-966c-b65fff5a87d8",
          "type": "State_Transition",
          "name": "sonando_salida",
          "sourceId": "b8f1e05f-4825-45c6-9d55-04462b00ff89",
          "targetId": "8431e33c-d9b5-4661-abbe-c97c30a99b46",
          "points": [],
          "min": 1,
          "max": 9999999,
          "properties": []
        },
        {
          "id": "a5053c54-ce1c-42ff-a7c2-27e8ff62a25a",
          "type": "Transition_State",
          "name": "salida_inactivo",
          "sourceId": "8431e33c-d9b5-4661-abbe-c97c30a99b46",
          "targetId": "5558d9e3-cfc2-40e8-be64-d08e54e217af",
          "points": [],
          "min": 1,
          "max": 1,
```

```
            "properties": []
          },
          {
            "id": "afd90e63-6f1d-4071-951b-389066f4349c",
            "type": "State_Transition",
            "name": "hablando_Transition 1",
            "sourceId": "a373c397-d72f-4472-8d68-dcbd7a35c1fd",
            "targetId": "190c92a3-2cbd-4ffa-9c86-0c395ff6053e",
            "points": [],
            "min": 1,
            "max": 9999999,
            "properties": []
          },
          {
            "id": "391a5de0-342d-4680-8f61-47d16f77ceb3",
            "type": "Transition_State",
            "name": "Transition 1_inactivo",
            "sourceId": "190c92a3-2cbd-4ffa-9c86-0c395ff6053e",
            "targetId": "5558d9e3-cfc2-40e8-be64-d08e54e217af",
            "points": [],
            "min": 1,
            "max": 1,
            "properties": []
          }
        ],
        "constraints": ""
      }
    ],
    "languagesAllowed": []
  },
  "applicationEngineering": {
    "models": [],
    "languagesAllowed": [],
    "applications": []
  }
 }
]
```