

Extendiendo HLVL para la especificación y análisis de modelos de variabilidad fraccionados

Juan Diego Carvajal Castaño

Sara Ortiz Drada

Universidad Icesi
Facultad de Ingeniería
Ingeniería de Sistemas
Cali
2021

Extendiendo HLVL para la especificación y análisis de modelos de variabilidad fraccionados

Proyecto de Grado

Presentado por
Juan Diego Carvajal Castaño
Sara Ortiz Drada

Dirigido por
Ángela Villota Gómez
Juan Manuel Reyes García

Universidad Icesi
Facultad de Ingeniería
Ingeniería de Sistemas
Cali
2021

Resumen

HLVL es un lenguaje de modelado de variabilidad que puede ser usado como un estándar para modelar la variabilidad de líneas de productos, ya que abarca la expresividad de distintos lenguajes para el modelado. Sin embargo, HLVL ofrece un soporte limitado para la escalabilidad de modelos de variabilidad, ya que no permite la especificación ni el análisis de modelos de variabilidad fraccionados.

El objetivo de este proyecto es desarrollar mejoras para el soporte de escalabilidad de HLVL, para la especificación y análisis de modelos de variabilidad fraccionados, es decir, modelos que impliquen un gran número de elementos y/o necesiten reflejar las preocupaciones de distintos *stakeholders*. Para lograrlo, se proponen los siguientes objetivos específicos: (i) seleccionar un conjunto de técnicas y/u operaciones que permitan soportar la escalabilidad de modelos especificados en HLVL; (ii) desarrollar una extensión del lenguaje HLVL que permita soportar la escalabilidad de modelos y (iii) evaluar el componente de modelado y análisis de la extensión.

Para el desarrollo de este proyecto se utilizó *Design Science*, un paradigma usado para hacer investigación en ingeniería de *software* y sistemas de información. Su objetivo es la creación de artefactos innovadores que sirvan como solución a problemas reales en los que se involucran personas, organizaciones y sistemas técnicos. El proceso de *Design Science* conlleva los siguientes pasos: Identificación del Problema y Motivación, Objetivos de la solución, Diseño y Desarrollo, Demostración, Evaluación, Comunicación.

Los siguientes son los resultados del proyecto:

- Reporte de escenarios que definen las operaciones y técnicas para el soporte de escalabilidad de lenguajes de modelado de variabilidad.
- Diseño de prototipos para las operaciones y técnicas para el soporte de escalabilidad.
- Documento del diseño de la extensión del lenguaje.
- Repositorios con la implementación de los componentes de modelado y análisis de la extensión.
- Evaluación de los componentes de modelado y análisis de la extensión.

Palabras claves: Líneas de productos de software, Modelado de Variabilidad, Análisis de Variabilidad, Lenguajes de Variabilidad, Escalabilidad, HLVL, Modelos de Variabilidad Fraccionados.

Abstract

HLVL is a variability modeling language that can be used as a standard for modeling the variability of product lines, as it encompasses the expressiveness of different modeling languages. However, HVLV offers limited support for variability model scalability, as it does not allow the specification or analysis of fractional variability models.

The objective of this project is to develop improvements for the scalability support of HVLV, for the specification and analysis of fractional variability models, that is, models that involve a large number of elements and / or need to reflect the concerns of different stakeholders. To achieve this, the following specific objectives are proposed: (i) select a set of techniques and / or operations that allow scalability of models specified in HVLV; (ii) develop an extension to the HVLV language to support model scalability and (iii) evaluate the modeling and analysis component of the extension.

For the development of this project, Design Science was used, a paradigm used to do research in software engineering and information systems. Its objective is the creation of innovative artifacts that serve as a solution to real problems in which people, organizations and technical systems are involved. The Design Science process involves the following steps: Identification of the Problem and Motivation, Objectives of the solution, Design and Development, Demonstration, Evaluation, Communication.

The following are the results of the project:

- Report of scenarios that define the operations and techniques for the scalability support of variability modeling languages.
- Prototype design for the operations and techniques to support scalability.
- Language extension design document.
- Repositories with the implementation of the modeling and analysis components of the extension.
- Evaluation of the extension modeling and analysis components.

Keywords: Software Product Lines, Variability Modeling, Variability Analysis, Variability Languages, Scalability, HVLV, Fractional Variability Models.

Tabla de contenido

CAPÍTULO 1: Motivación y Antecedentes, Formulación del Problema y Objetivos	1
1. Motivación y Antecedentes	1
1.1. Contexto	1
1.2. Antecedentes del problema	3
1.3. Justificación	4
2. Formulación del Problema	6
3. Objetivos	7
3.1. Objetivo general	7
3.2. Objetivos específicos	7
4. Metodología	8
4.1. Esquema de trabajo	9
4.2. Fases de desarrollo del proyecto	11
CAPÍTULO 2: Marco Teórico y Estado del Arte	14
1. Marco Teórico	14
1.1. Ingeniería de Líneas de Productos De Software	14
1.2. Modelado de Variabilidad	14
1.3. Modelos de Variabilidad Fraccionados	15
1.4. Configuración	16
1.5. Análisis Automático	16
1.6. Escalabilidad	23
1.7. High-Level Variability Language (HLVL)	24
2. Estado del Arte	26
2.1. Lenguajes	26
2.2. Análisis Comparativo	28
CAPÍTULO 3: Escenarios para el Soporte de Escalabilidad de HLVL	30
1. Mecanismos de Modelado	31
1.1. Composición a través de Herencia	32

1.2. Composición a través de Agregación.....	33
1.3. Composición a través de <i>Include</i>	34
1.4. Composición a través de <i>Import</i>	35
1.5. Composición y modularización través de Superposición	36
1.6. Modularización través de interfaces	37
2. Operaciones de Análisis	39
2.1. Composición a través de <i>merge</i>	39
2.2. Composición a través de agregación	42
3. Criterios de Agrupación	44
4. Escenarios	46
5. Niveles	47
6. Ruta de implementación.....	48
CAPÍTULO 4: Diseño de la Extensión	51
1. Herencia.....	51
1.1. Sintaxis.....	51
1.2. Semántica.....	52
1.3. Lógica de transformación	53
2. Operación de Agregación	54
2.1. Diseño	54
2.2. Implementación.....	56
3. Operación de Merge	59
3.1. Diseño	59
3.2. Implementación.....	60
CAPÍTULO 5: Evaluación de la Extensión.....	62
1. Contexto	62
2. Componente de modelado	63
2.1. Planeación.....	63
2.2. Análisis y resultados.....	64

2.3. Conclusiones.....	68
3. Componente de análisis.....	69
3.1. Planeación.....	69
3.2. Análisis y resultados.....	74
3.3. Conclusiones.....	74
CAPÍTULO 6: Conclusiones y Trabajo a Futuro	76
1. Resultados Obtenidos	76
1.1. Aportes relacionados con el objeto del proyecto	76
1.2. Aportes relacionados con el desarrollo de capacidades del investigador	77
1.3. Resultados y entregables	77
2. Conclusiones.....	79
Anexos	82
Lista de Acrónimos.....	82
Glosario de Términos	83
Análisis de participación	85
Árbol del Problema	86
Árbol de Objetivos	87
Propuesta de Arquitectura	88
Encuesta Soporte de Escalabilidad para HLVL.....	89
Preguntas <i>mecanismo Herencia</i>	89
Preguntas <i>mecanismo Interfaces</i>	90
Preguntas Consentimiento	91
Análisis de riesgos y limitaciones.....	92
Referencias Bibliográficas	93

Índice de Ilustraciones

Ilustración 1. Modelo de variabilidad de una línea de productos de teléfonos móviles.	2
Ilustración 2. Proceso de investigación aplicada de <i>Design Science</i>	8
Ilustración 3. Proceso de transformación de un modelo a su representación en CNF formato DIMACS.	18
Ilustración 8. Ejemplo de un script en HLVL.....	24
Ilustración 9. Sintaxis en HLVL del modelo Procesador.	31
Ilustración 10. Sintaxis en HLVL para el modelo Sistema Operativo.	31
Ilustración 11. Propuesta de sintaxis en HLVL de la composición a través de Herencia....	33
Ilustración 12. Propuesta de sintaxis en HLVL de la composición a través de agregación.	34
Ilustración 13. Archivo que contiene únicamente los elementos del modelo PC	34
Ilustración 14. Propuesta de sintaxis en HLVL de la composición a través de include.	35
Ilustración 15. Propuesta de sintaxis en HLVL de la composición a través de import.	36
Ilustración 16. Propuesta de sintaxis en HLVL de la modularización y composición a través de Superposición.....	36
Ilustración 17. Propuesta de sintaxis en HLVL de la modularización a través de interfaces	37
Ilustración 18. Propuesta de sintaxis en HLVL del uso de interfaces.....	38
Ilustración 19. Ejemplo gráfico del funcionamiento de la operación de Merge en modo union.....	39
Ilustración 20. Ejemplo gráfico del funcionamiento de la operación de <i>Merge</i> en modo intersección	40
Ilustración 21. Ejemplo gráfico del funcionamiento de la operación de <i>Merge</i> en modo diferencia.....	41
Ilustración 22. Ejemplo de agregación gráfico del funcionamiento de la operación de agregación.....	43
Ilustración 23. Diagrama de dependencias y ruta de implementación de escenarios.....	49
Ilustración 24. Declaración modelos heredados, extensión HLVL.....	52

Ilustración 25. Referenciación de elementos de modelos heredados, extensión HLVL	52
Ilustración 26. Pseudocódigo operación de Agregación.....	54
Ilustración 27. Pseudocódigo método addElements.....	55
Ilustración 28. Pseudocódigo método addRelations.....	56
Ilustración 29. Diagrama de secuencia de la operación de Agregación.....	57
Ilustración 30. Pseudocódigo operación de Merge.....	59
Ilustración 31. Diagrama de secuencia de la operación de <i>Merge</i>	60
Ilustración 32. Pregunta 1 sección Herencia, encuesta soporte HLVL.....	64
Ilustración 33. Pregunta 2 sección Herencia, encuesta soporte HLVL.....	65
Ilustración 34. Pregunta 1 sección interfaces, encuesta soporte HLVL.....	66
Ilustración 35. Pregunta 2 sección interfaces, encuesta soporte HLVL.....	67
Ilustración 36. Caso de estudio <i>Toy Model</i>	71
Ilustración 37. Caso de estudio FameDB	71
Ilustración 38. Caso de estudio GPL	72
Ilustración 39. Caso de estudio <i>CarSystem</i>	73
Ilustración 40. Árbol del problema	86
Ilustración 41. Árbol de objetivos.....	87

Índice de Tablas

Tabla 1. Roles y funciones de los integrantes.....	10
Tabla 2. Relación entre el objetivo específico 1, fases, actividades y entregables del proyecto.....	11
Tabla 3. Relación entre el objetivo específico 2, fases, actividades y entregables del proyecto.....	12
Tabla 4. Relación entre el objetivo específico 3, fases, actividades y entregables del proyecto.....	13
Tabla 5. Transformación de modelos de características a lógica proposicional.....	18
Tabla 6. Transformación de lógica proposicional a CNF.	20
Tabla 7. Transformación de lógica proposicional a CNF, ejemplo Teléfono Móvil.....	21
Tabla 8. Mapeo de características a identificadores numéricos.....	22
Tabla 9. Transformación de CNF a formato Dimacs, ejemplo Teléfono Móvil.	22
Tabla 10. Comparación de alternativas de solución para el problema	28
Tabla 11. Clasificación de mecanismos de modelado y operaciones de análisis.	45
Tabla 12. Escenarios de implementación para el soporte de escalabilidad de HLVL.....	46
Tabla 13. Niveles de dificultad de los escenarios según su implementación.....	47
Tabla 14. Resultados evaluación Agregación.....	74
Tabla 15. Resultados evaluación Merge.....	74
Tabla 16. Matriz de riesgos.....	92

CAPÍTULO 1: Motivación y Antecedentes, Formulación del Problema y Objetivos

1. Motivación y Antecedentes

1.1. CONTEXTO

Las estrategias de reutilización hacen parte de la ingeniería de *software* desde sus comienzos. La reutilización se introduce en 1968 en la conferencia de ingeniería de *software* de la OTAN, y se define como la creación de sistemas de *software* a partir de *software* existente en lugar de crear sistemas desde cero (Krueger, 1992). La reutilización se convirtió en uno de los focos para la construcción de *software* de calidad ya que permite construir mejor *software*. Los beneficios de la reutilización son numerosos, por ejemplo, como la reducción de tiempos, la simplificación en el desarrollo de *software*, la mejora de la calidad, y la reducción de costos y mayor mantenibilidad. No obstante, para obtener beneficios es necesario que la estrategia de reutilización sea sistemática.

La reutilización sistemática de *software* da origen a lo que se conoce como *Software Product Lines Engineering* (SPLE), una rama de la ingeniería de *software* dedicada a la producción de líneas de productos o familias de productos. Una línea de productos de *software* (SPL) es un conjunto de sistemas intensivos de *software* que comparten características comunes y que satisfacen las necesidades específicas de un segmento del mercado en particular (P. Clements, L. Northrop, 2000).

Las líneas de productos son especificadas por medio de modelos de variabilidad. Un modelo de variabilidad es una representación que describe los elementos comunes y variables de una línea de productos (Berger et al., 2013). La importancia de los modelos de variabilidad reside en su capacidad de impactar una gran variedad de actividades a lo largo del desarrollo de las líneas de productos. Los modelos de variabilidad impactan en

diferentes niveles de abstracción, desde la ingeniería de requerimientos, hasta la codificación y despliegue (Galindo et al., 2019).

En la Ilustración 1 se muestra un ejemplo de modelo de variabilidad escrito en el lenguaje de *FeatureIDE* traducido de (Benavides et al., 2010). Este modelo representa una línea de productos de teléfonos móviles y muestra las características que puede tener el teléfono y las restricciones para derivar productos específicos. Por ejemplo, un teléfono tiene un tipo de pantalla, pero podría tener una cámara, MP3 o ambas.

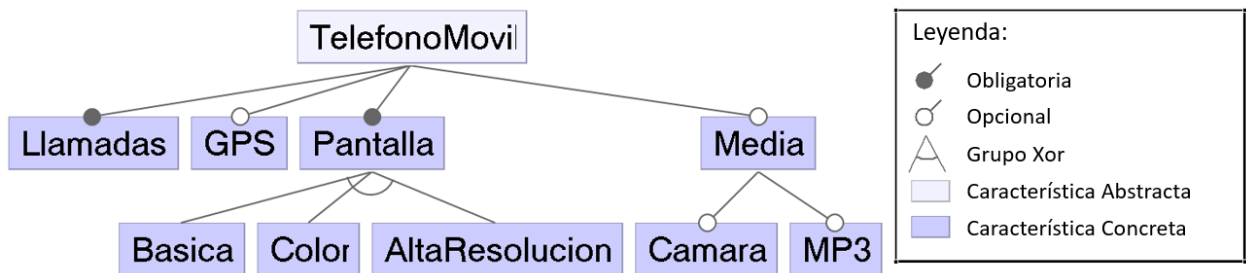


Ilustración 1. Modelo de variabilidad de una línea de productos de teléfonos móviles (Benavides et al., 2010).

Ejemplos de productos válidos para esta línea de productos son: un teléfono con pantalla de alta resolución, llamadas y *gps*; un teléfono con pantalla básica, llamadas, *gps* y MP3; un teléfono con pantalla a color, llamadas y cámara. De este modelo pueden ser derivados 24 productos diferentes.

Como los modelos de variabilidad son plantillas para la construcción de productos concretos, es importante que (i) puedan reflejar las necesidades de los *stakeholders*, (ii) sean fáciles de especificar, (iii) sean fáciles de mantener y (iv) no tengan defectos. Para garantizar estas características, las herramientas que asisten la especificación de modelos de variabilidad están provistas de operaciones que permiten extraer información de dichos modelos. Estas operaciones son conocidas como operaciones de análisis (Galindo et al., 2019). Las operaciones de análisis permiten responder a preguntas como: ¿el modelo está bien hecho?, ¿qué errores tiene?, ¿cómo se puede corregir?, ¿cuántos productos pueden ser derivados del modelo?

En una línea de productos con una gran cantidad de elementos puede ser impráctico el uso de un único modelo de variabilidad para su representación. Una alternativa es el uso de modelos de variabilidad fraccionados, en la cual diferentes modelos son usados para representar diferentes aspectos o vistas sobre una línea de productos (Chavarriaga, 2017). El uso de modelos fraccionados permite mejorar el proceso de reutilización, especificación y análisis de sistemas complejos que reflejan de forma independiente las necesidades de distintos *stakeholders*.

1.2. ANTECEDENTES DEL PROBLEMA

High-Level Variability Language (HLVL) es un lenguaje de modelado de variabilidad expresivo y flexible, que puede ser usado como un lenguaje estándar para modelar la variabilidad en líneas de productos y que posee una sintaxis formalmente definida que se asemeja a los lenguajes de programación (Villota et al., 2019). Por lo tanto, es una propuesta de lenguaje estándar que permite el modelamiento de sistemas que requieren diferentes niveles de expresividad.

Al hacer uso de HLVL, se requiere la especificación y análisis de modelos fraccionados dado que actualmente sólo soporta el manejo de modelos independientes. Esto significa que, si se quisiera modelar un sistema con un gran número de elementos, se tendría que especificar en un sólo modelo, lo que dificulta el modelado y entendimiento de este. Esta situación también complica la especificación de sistemas en los que es necesario el modelado de distintos aspectos que reflejen las necesidades de diferentes *stakeholders*, ya que implica la coordinación de un conjunto de personas para que trabajen bajo un mismo modelo. Los dos casos anteriores son las causas de la complejidad de un sistema, al momento de modelar su variabilidad.

Realizar el análisis de un sistema es un proceso que escala en dificultad de acuerdo con la complejidad del sistema. En la industria los modelos de variabilidad que representan una familia de productos pueden llegar a los cientos de puntos de variación o características variables. Por ejemplo, un modelo de variabilidad que especifica la variabilidad en una

línea de productos puede llegar a tener alrededor de 300 subsistemas diferentes (Chavarriaga, 2017), cada uno podría verse reflejado en un modelo independiente que simplifique el esfuerzo del diseñador y que mediante algún tipo de especificación del lenguaje que permita su posterior análisis.

1.3. JUSTIFICACIÓN

El objetivo de la gestión de la variabilidad, de la cual el modelado y análisis son actividades claves, es maximizar el retorno de inversión de los procesos de construcción y mantenimiento de líneas de productos a través del tiempo (Bachmann & Clements, 2005). Por tanto, es poca la utilidad del modelamiento si lo único que representa para la organización es un conjunto de archivos o dibujos en un computador. Es por esto que el análisis de modelos de variabilidad se vuelve un proceso fundamental dentro de este contexto, ya que es un medio para obtener información valiosa sobre una línea o familia de productos que le permita a la organización tomar decisiones que le ahorren tiempo y dinero.

Debido a la naturaleza del sistema, en el modelamiento es común que se presenten tres problemas que hacen del modelado un proceso complicado: (i) diferentes *stakeholders* requieren modelar diferentes aspectos del sistema; (ii) el sistema es demasiado grande para ser especificado en un solo modelo; (iii) dado que no existe un lenguaje de modelado estándar, diferentes partes del modelo pueden estar especificadas en lenguajes distintos.

Una propuesta de solución a los problemas (i) y (ii) es el uso de modelos de variabilidad fraccionados o modulares (Chavarriaga, 2017) los cuales son usados para representar diferentes vistas o dominios de un mismo sistema, producto de la existencia de diferentes *stakeholders*. Así mismo, este enfoque resulta útil para el modelamiento de sistemas con una gran cantidad de elementos. Si bien el problema está parcialmente resuelto en cuanto al modelamiento, para el proceso de análisis es necesario hacer uso de mecanismos que permitan analizar el sistema como un único modelo. Estos mecanismos, permiten hacer unión de modelos y por tanto el análisis de un sistema especificado con modelos de variabilidad fraccionados, solucionando los problemas (i) y (ii).

La situación con el problema (iii) tiene que ver con la inexistencia de un lenguaje estándar para el modelado de la variabilidad. Existe una propuesta de estándar, el lenguaje HLVL, que abarca la expresividad de una gran variedad de lenguajes (Villota et al., 2019), atacando el último problema. Sin embargo, HLVL no permite la especificación de modelos fraccionados, por lo tanto, tampoco permite su análisis.

2. Formulación del Problema

Una de las limitaciones de HLVL es que ofrece un soporte limitado para la escalabilidad de modelos de variabilidad, por dos razones principales. El lenguaje no permite la especificación ni el análisis de modelos de variabilidad fraccionados. Debido a estas razones HLVL presenta limitaciones en las siguientes actividades para el manejo de líneas de productos: (i) modelar de sistemas con un gran número de elementos; (ii) realizar el análisis de un sistema especificado en modelos de variabilidad fraccionados; y (iii) modelar sistemas que requieren que el modelo refleje las preocupaciones de diferentes *stakeholders*. Por lo tanto, estas limitaciones hacen que la aplicación de HLVL para la gestión de la variabilidad de líneas de productos complejas no sea viable.

3. Objetivos

3.1. OBJETIVO GENERAL

Desarrollar mejoras para soportar la escalabilidad de HLVL para la especificación y análisis de modelos de variabilidad con un gran número de elementos y/o que necesitan reflejar las preocupaciones de diferentes *stakeholders*.

3.2. OBJETIVOS ESPECÍFICOS

1. Seleccionar un conjunto de técnicas y/u operaciones que permitan soportar la escalabilidad de modelos especificados en HLVL.
2. Desarrollar una extensión del lenguaje HLVL que permita soportar la escalabilidad de modelos.
3. Evaluar los componentes de modelado y análisis de la extensión.

4. Metodología

Para el desarrollo de este proyecto se utilizó *Design Science*, un paradigma usado para hacer investigación en ingeniería de *software* y sistemas de información. Su objetivo es la creación de artefactos innovadores que sirvan como solución a problemas reales en los que se involucran personas, organizaciones y sistemas técnicos (Hevner, 2004).

En la Ilustración 2 se especifica el proceso de investigación aplicada de *Design Science*, el cual define seis pasos para el desarrollo de la investigación. El investigador tiene la libertad de dar inicio al proceso en cualquiera de los pasos e iterar sobre ellos, aun cuando el proceso está estructurado según un orden secuencial (Peffer et al., 2006).

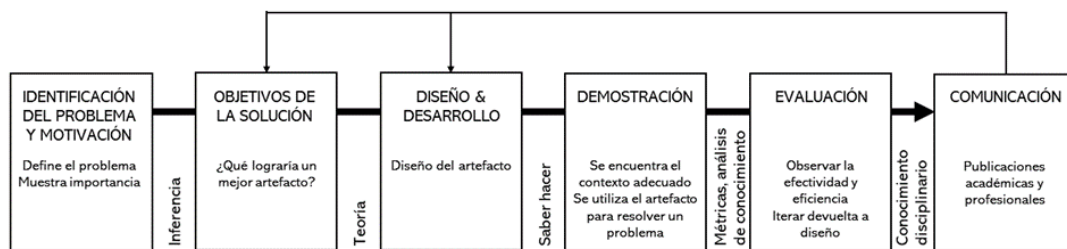


Ilustración 2. Proceso de investigación aplicada de *Design Science* (Peffer et al., 2006).

Design Science describe los pasos del proceso de la siguiente manera (Peffer et al., 2006):

- 1. Identificación del problema y motivación:** se define el problema específico de investigación de forma que capture su complejidad y se justifica el valor de la solución.
- 2. Objetivos de la solución:** se infieren los objetivos de la solución en términos de cómo la solución deseada será mejor que las soluciones actuales.
- 3. Diseño y Desarrollo:** se determinan las funcionalidades y arquitectura de los artefactos y se lleva a cabo su implementación.
- 4. Demostración:** se demuestra la eficacia del artefacto para resolver el problema. Esto puede incluir actividades como: experimentación, simulación, casos de estudio, etc.

5. **Evaluación:** se observa y mide el nivel de eficiencia del artefacto en cuanto a la solución del problema. Esto puede incluir la comparación de las funcionalidades del artefacto con los objetivos de la solución o con otras soluciones propuestas. Al final de este paso, se decide si se vuelve a iterar sobre el paso tres.
6. **Comunicación:** se comunica la importancia del problema y la utilidad, novedad y efectividad del artefacto a audiencias relevantes.

Design Science es el método que mejor se ajusta al proyecto por las siguientes razones:

1. *Design Science* es especial para hacer investigaciones en el área de ingeniería de *software* y sistemas de información (Peppers et al., 2006).
2. Gracias a la planificación y desarrollo iterativo, se puede realizar una mejor evaluación de los riesgos que pueden llegar a introducirse durante la ejecución del proceso.
3. Se cuenta con un equipo de desarrollo pequeño (2 integrantes).

4.1. ESQUEMA DE TRABAJO

El esquema de trabajo está distribuido de la siguiente manera:

Junio-Julio:

- 13 horas de trabajo semanales para el desarrollo del proyecto.
- Una reunión semanal de aproximadamente una hora y media con los tutores del proyecto. El objetivo de esta reunión es la resolución de dudas y validación de las actividades realizadas.

Agosto-Noviembre:

- 10 horas de trabajo semanales para el desarrollo del proyecto.
- Una reunión semanal de aproximadamente una hora con los tutores del proyecto. El objetivo de esta reunión es la resolución de dudas y validación de las actividades realizadas.
- El tiempo definido para entregar retroalimentaciones será de dos semanas a partir de la entrega a los tutores.

En la Tabla 1 se especifican los roles y funciones que desempeñan cada uno de los integrantes dentro del proyecto.

Tabla 1. Roles y funciones de los integrantes

Rol	Integrante	Funciones
Investigador	Juan Diego Carvajal Castaño	Planeación y desarrollo del proyecto en su totalidad
	Sara Ortiz Drada	
Tutor	Ángela Villota	Orientar a los estudiantes en la elaboración del proyecto
	Juan Manuel Reyes	Supervisar el desarrollo del proyecto y la escritura de los resultados Validar el contenido técnico del proyecto

4.2. FASES DE DESARROLLO DEL PROYECTO

En las Tablas Tabla 2, Tabla 3 y Tabla 4 se especifican las actividades correspondientes al desarrollo de cada uno de los objetivos específicos de acuerdo con los diferentes pasos del proceso de *Design Science*. Además, se especifican los respectivos entregables para cada conjunto de actividades.

Tabla 2. Relación entre el objetivo específico 1, fases, actividades y entregables del proyecto

Objetivo Específico	Fases	Actividades	Entregables
OE 1: Seleccionar un conjunto de técnicas y/u operaciones que permitan soportar la escalabilidad de modelos de variabilidad fraccionados.	Identificación del Problema y Motivación	Definir las características de escalabilidad que se implementarán en HLVL, por medio de escenarios	Sección del documento del Proyecto (CAPÍTULO 3: Escenarios para el Soporte de Escalabilidad de HLVL) Repositorio del código de los prototipos
		Caracterizar las implementaciones de escalabilidad (características: composición, modularidad y evolución)	
	Diseño y Desarrollo	Investigar herramienta XTEXT	
		Definir los criterios de comparación de las implementaciones para cada característica	
		Diseñar prototipos de las implementaciones	
		Implementar los prototipos	
		Comparar los prototipos según los criterios definidos	
	Demostración	Probar los prototipos con ejemplos pequeños	
	Evaluación	Seleccionar la(s) mejor(es) implementación(es) por cada característica	
	Comunicación	Incorporar el reporte al documento del Proyecto	
		Incorporar resultados a la presentación oral del Proyecto	Diapositivas de la presentación del Proyecto

Tabla 3. Relación entre el objetivo específico 2, fases, actividades y entregables del proyecto

Objetivo Específico	Fases	Actividades	Entregables
OE 2: Desarrollar una extensión del lenguaje HLVL que permita soportar la escalabilidad de modelos.	Diseño y Desarrollo	Especificar la sintaxis de la extensión del lenguaje HLVL	Capítulo del diseño de la extensión (CAPÍTULO 4: Diseño de la Extensión)
		Especificar la semántica de la extensión del lenguaje HLVL	
		Especificar la función de transformación de la extensión del lenguaje HLVL a programación de restricciones	
		Implementar la extensión del lenguaje HLVL	Repositorio con el código de la extensión
	Demostración	Probar la extensión con ejemplos pequeños	Repositorio con el código de las pruebas
	Evaluación	Validar el nivel de cubrimiento en amplitud de los escenarios	
		Validar el nivel de cubrimiento en profundidad de los escenarios	
	Comunicación	Incorporar entregables al documento del Proyecto	Diapositivas de la presentación del Proyecto
		Incorporar entregables a la presentación oral del Proyecto	

Tabla 4. Relación entre el objetivo específico 3, fases, actividades y entregables del proyecto

Objetivo Específico	Fases	Actividades	Entregables
OE3: Evaluar los componentes de modelado y análisis de la extensión	Diseño y Desarrollo	Diseñar el formulario de la encuesta	Formulario encuesta
		Diseño y desarrollo de pruebas con modelos pequeños, medianos y grandes para las operaciones de Agregación y <i>Merge</i>	Repositorio con las pruebas y modelos en HLVL, FAMILIAR
	Demostración	Realizar encuesta a expertos	Sección del documento del Proyecto (¡Error! No se encuentra el origen de la referencia.)
	Evaluación	Analizar resultados	
		Obtener conclusiones	
	Comunicación	Elaborar un reporte	Diapositivas de la presentación del Proyecto
		Incorporar resultados al documento del Proyecto de Grado	
		Incorporar entregables a la presentación oral del Proyecto	

CAPÍTULO 2: Marco Teórico y Estado del Arte

1. Marco Teórico

1.1. INGENIERÍA DE LÍNEAS DE PRODUCTOS DE SOFTWARE

La Ingeniería de Líneas de Productos de *Software* (SPLE por sus siglas en inglés) se define como el paradigma para desarrollar aplicaciones de *software* utilizando plataformas, es decir, cualquier base tecnológica en la cual se construyen o procesan otras tecnologías y personalización masiva, que es la producción a gran escala de bienes adaptados a las necesidades de los clientes (Pohl et al., 2005).

La importancia que tiene la Ingeniería de Líneas de Productos de *Software* radica en los beneficios que implica hacer gestión sobre los elementos comunes y variables de muchos productos: los artefactos reutilizados en diferentes sistemas llevan a una reducción de costos y tiempos de desarrollo y debido a que los artefactos comunes son revisados y probados en varios productos, se aumenta la posibilidad de detección y corrección de fallas y, por lo tanto, se asegura un aumento de la calidad de los productos.

1.2. MODELADO DE VARIABILIDAD

La variabilidad es un concepto que aparece al momento de administrar un conjunto de productos, entre los cuáles se identifican elementos comunes y variables. El término variabilidad se refiere a capacidad o tendencia de algo a cambiar, dentro de la ingeniería de productos de *software* se refiere a los cambios que son introducidos dentro de un sistema, para producir sistemas diferentes (Pohl et al., 2005).

Los lenguajes para el modelado de variabilidad proveen un conjunto de constructos los cuales permiten especificar los elementos que son variables dentro de una línea de productos de *software*, las diferentes variantes de esos elementos y establecer un conjunto de restricciones sobre dichas variantes que especifican las reglas a seguir al momento de combinarlas dentro de una configuración o producto derivado.

Un concepto fundamental para el modelado de variabilidad es el de unidad de variabilidad. Las unidades de variabilidad son la representación de los elementos variables del sistema, dentro de cada lenguaje. Algunos ejemplos de unidades de variabilidad son: las características, para los modelos orientados a características; los puntos de variación, para los modelos ortogonales de variabilidad y las decisiones, para modelos orientados a decisiones. Las unidades de variabilidad tienen distintos tipos de acuerdo con la cantidad de variantes que representan. Son booleanas cuando sus variantes están asociadas únicamente a dos opciones, serán no booleanas en el caso contrario (Villota et al., 2019).

Ciertas combinaciones de variantes pueden resultar en productos no válidos para llevar a producción. Para controlar estas contradicciones en los modelos, los lenguajes permiten especificar relaciones entre los elementos variables, las cuales funcionan como restricciones sobre la configuración de productos del modelo. Las posibles relaciones que se pueden dar entre ellas son (Villota et al., 2019):

- **Inclusión/Exclusión:** Entre los elementos variables (A) y (B) se pueden presentar: la inclusión, si la aparición de (A) en una configuración dada, implica obligatoriamente la aparición de B y la exclusión, si (A) y (B) no pueden aparecer ambas en ninguna configuración válida del modelo (Villota et al., 2019).
- **Descomposición:** Esta es un tipo de relación jerárquica de padre-hijo(s), en la que un elemento variable padre puede estar asociado a un elemento variable o variante hijo de forma uno-a-uno o a un conjunto de elementos o variantes hijos de forma uno-a-muchos. En el caso de la descomposición uno-a-muchos, se debe especificar el valor mínimo y máximo de hijos que pueden estar presentes en una configuración cuando su padre está incluido (Villota et al., 2019)
- **Expresiones de restricción:** Algunos lenguajes permiten incorporar restricciones complejas en el modelo por medio de expresiones compuestas por operadores lógicos, relacionales, aritméticos, etc. Estas expresiones permiten incorporar reglas contextuales del dominio de aplicación (Villota et al., 2019).

1.3. MODELOS DE VARIABILIDAD FRACCIONADOS

Las líneas de productos de *software* son especificadas mediante modelos de variabilidad. En el caso de sistemas pequeños o simples, el modelamiento usualmente es monolítico,

es decir, en un único modelo. Sin embargo, existen dos factores que hacen que el modelado monolítico pierda practicidad. El primer factor es la cantidad de elementos del sistema, ya que implica especificar una cantidad muy grande de elementos dentro de un único modelo. El segundo factor es la existencia de diferentes *stakeholders*, los cuales están familiarizados con diferentes dominios que aportan variabilidad al sistema. Estos dos factores dificultan considerablemente la interpretación y modelado, ya que siguiendo el modelado monolítico los modelos resultantes serían demasiado extensos y/o mezclarían diferentes dominios de aplicación del sistema.

Las problemáticas mencionadas anteriormente llevaron a la creación de un enfoque modular alternativo en el cual se utilizan modelos de variabilidad fraccionados para representar sistemas complejos. Los modelos de variabilidad fraccionados son utilizados para representar diferentes subsistemas o vistas de un mismo modelo en diferentes modelos (Chavarriaga, 2017). Como resultado, se tienen modelos mucho más pequeños y fáciles de entender, revisar y reutilizar.

1.4. CONFIGURACIÓN

Cuando se tiene un modelo de variabilidad sobre una línea de productos, el proceso con el cual se derivan productos concretos del modelo se conoce como configuración. Al momento de obtener una configuración, se van agregando elementos del modelo, lo cual ocasiona que las relaciones entre los elementos agregados limiten la visibilidad y disponibilidad de los demás elementos. Para que una configuración sea válida, debe cumplir con las restricciones propias del modelo y de sus elementos (Asadi et al., 2014).

1.5. ANÁLISIS AUTOMÁTICO

La complejidad de una familia de productos puede ser tan alta, que su modelamiento implique la construcción de modelos de variabilidad con cientos o miles de características. Es por esto, que se han propuesto un conjunto de operaciones que permitan detectar problemas y extraer información importante de forma automática sobre este tipo de modelos. (Benavides et al., 2010).

Para implementar las operaciones de análisis sobre modelos de variabilidad se han propuesto diversos enfoques, los cuales siguen una serie de pasos para llevar a cabo la implementación: (i) traducir los modelos de variabilidad a una representación formal, (ii) analizar la representación alternativa utilizando *solvers*, y (iii) presentar los resultados traduciendo la respuesta del *solver* a los elementos del modelo (Benavides et al., 2010). Existen cuatro categorías de enfoques para implementar las operaciones de análisis sobre modelos de variabilidad (Benavides et al., 2010): (i) los que traducen modelos de variabilidad a Lógica Proposicional y utilizan *solvers* de problemas de satisfacibilidad booleana (SAT) o de Diagramas Binarios de Decisión (BDD); (ii) los que traducen de modelos a problemas de satisfacción de restricciones; (iii) los que utilizan Descripción Lógica, y (iv) los que utilizan algoritmos *ad-hoc* para el análisis (Benavides et al., 2010).

1.5.1. Proceso de Transformación lógica de modelos

Uno de los enfoques para el soporte automatizado de las operaciones de análisis de modelos de variabilidad, es el uso de la lógica proposicional que representa dichos modelos como un conjunto de proposiciones lógicas. El análisis basado en lógica proposicional consiste en tomar la representación lógica o fórmula lógica de un modelo y pasarla a un *software* llamado SAT *solver*, el cual verifica si dicha fórmula es satisfacible, es decir, si es posible que al evaluarla de acuerdo a alguna combinación de valores pueda resultar verdadera (Benavides et al., 2010).

En la Ilustración 3 se muestra el proceso de transformación necesario para analizar un modelo de variabilidad por medio de un SAT *solver*, este proceso consta de los siguientes pasos: primero, transformar el modelo a Transformación del **Modelo a Lógica Proposicional**, es decir, representarlo mediante un conjunto de proposiciones lógicas; segundo, modificar dichas proposiciones a través de equivalencias lógicas para cumplir con la Transformación de la **representación en Lógica Proposicional a Forma Normal Conjuntiva (CNF)** (CNF, por sus siglas en inglés); por último transformar la fórmula CNF a su representación en Forma Normal Conjuntiva – Formato DIMACS. El resultado de este último paso es la entrada que toma el SAT *solver* para analizar el modelo.

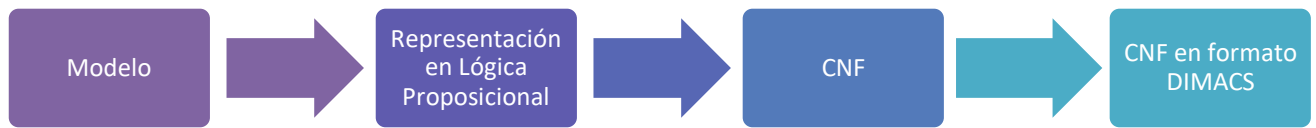


Ilustración 3. Proceso de transformación de un modelo a su representación en CNF formato DIMACS.

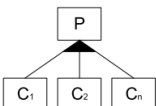
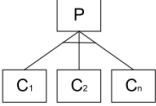
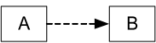

A continuación, se describe brevemente cada uno de los pasos del proceso.

Transformación del Modelo a Lógica Proposicional

Este paso consiste en la transformación de un modelo a su representación como fórmula proposicional. Una fórmula proposicional está formada por un conjunto de símbolos primitivos o variables de tipo booleano que representan las características de un modelo, y un conjunto de conectivos lógicos que restringen los valores de dichas variables, por ejemplo $\wedge, \vee, \rightarrow, \leftrightarrow$ (Benavides et al., 2010). Para representar un modelo de características en lógica proposicional se utilizan las equivalencias presentadas en la [Tabla 5](#). En la columna Relación se especifica la sintaxis de las relaciones del modelo, en la columna Lógica Proposicional se especifica un mapeo entre la relación del modelo y su representación lógica y en la columna Ejemplo Teléfono Móvil se muestra el resultado de la transformación del modelo Teléfono Móvil de la [Ilustración 1](#) a su representación en lógica proposicional.

Tabla 5. Transformación de modelos de características a lógica proposicional, tomado de (Benavides et al., 2010).

Relación		Transformación a Lógica Proposicional	Transformación Ejemplo Teléfono Móvil
Obligatorio		$P \leftrightarrow C$	<i>TelefonoMovil</i> \leftrightarrow <i>Llamadas</i> <i>TelefonoMovil</i> \leftrightarrow <i>Pantalla</i>
Opcional		$C \rightarrow P$	<i>GPS</i> \rightarrow <i>MobilePhone</i> <i>Media</i> \rightarrow <i>TelefonoMovil</i>

OR		$P \leftrightarrow (C_1 \vee C_2 \vee \dots \vee C_n)$	$Media \leftrightarrow (Camara \vee MP3)$
XOR		$(C_1 \leftrightarrow (\neg C_2 \wedge \dots \wedge \neg C_n \wedge P)) \wedge$ $(C_2 \leftrightarrow (\neg C_1 \wedge \dots \wedge \neg C_n \wedge P)) \wedge$ $(C_n \leftrightarrow (\neg C_1 \wedge \neg C_2 \wedge \dots \wedge \neg C_{n-1} \wedge P))$	$(Basico \leftrightarrow (\neg Color \wedge \neg AltaResolucion \wedge Pantalla)) \wedge$ $(Color \leftrightarrow (\neg Basico \wedge \neg AltaResolucion \wedge Pantalla)) \wedge$ $(AltaResolucion \leftrightarrow (\neg Basic \wedge \neg Color \wedge Pantalla))$
Implicación		$A \rightarrow B$	$Camara \rightarrow AltaResolucion$
Exclusión		$\neg(A \wedge B)$	$\neg(GPS \wedge Basico)$

Transformación de la representación en Lógica Proposicional a Forma Normal Conjuntiva (CNF)

La Forma Normal Conjuntiva es una forma estándar para representar las fórmulas proposicionales (Benavides et al., 2010). Esta representación consiste en una conjunción de cláusulas, donde cada cláusula es una disyunción de variables booleanas. La CNF es la forma más utilizada por los SAT *solvers*, en dónde sólo se permiten tres tipos de conectivos lógicos: \neg, \wedge, \vee . En la Tabla 6 se muestra el mapeo entre las proposiciones lógicas y su formato CNF y en la Tabla 7 se muestra el resultado de este proceso sobre el modelo Teléfono Móvil.

Tabla 6. Transformación de lógica proposicional a CNF.

Lógica Proposicional	Transformación a Forma Normal Conjuntiva (CNF)
$P \leftrightarrow C$	$(\neg P \vee C) \wedge (P \vee \neg C)$
$C \rightarrow P$	$\neg P \vee C$
$P \leftrightarrow (C_1 \vee C_2 \vee \dots \vee C_n)$	$(\neg P \vee C_1 \vee C_2 \vee \dots \vee C_n) \wedge (P \vee \neg C_1) \wedge (P \vee \neg C_2) \wedge \dots \wedge (P \vee \neg C_n)$
$(C_1 \leftrightarrow (\neg C_2 \wedge \dots \wedge \neg C_n \wedge P)) \wedge$ $(C_2 \leftrightarrow (\neg C_1 \wedge \dots \wedge \neg C_n \wedge P)) \wedge$ $(C_n \leftrightarrow (\neg C_1 \wedge \neg C_2 \wedge \dots \wedge \neg C_{n-1} \wedge P))$	$(\neg C_1 \vee \neg C_2) \wedge \dots \wedge (\neg C_1 \vee \neg C_n) \wedge (\neg C_1 \vee P) \wedge (C_1 \vee C_2 \vee \dots \vee C_n \vee \neg P) \wedge$ $(\neg C_2 \vee \neg C_1) \wedge \dots \wedge (\neg C_2 \vee \neg C_n) \wedge (\neg C_2 \vee P) \wedge (C_2 \vee C_1 \vee \dots \vee C_n \vee \neg P) \wedge$ $(\neg C_n \vee \neg C_1) \wedge (\neg C_n \vee \neg C_2) \wedge \dots \wedge (\neg C_n \vee \neg C_{n-1}) \wedge (\neg C_n \vee P) \wedge (C_n \vee C_1 \vee C_2$ $\vee \dots \vee C_{n-1} \vee \neg P)$
$A \rightarrow B$	$\neg A \vee \neg B$
$\neg(A \wedge B)$	$(\neg C_1 \vee \neg C_2) \wedge \dots \wedge (\neg C_1 \vee \neg C_n) \wedge (\neg C_1 \vee P) \wedge (C_1 \vee C_2 \vee \dots \vee C_n \vee \neg P) \wedge$ $(\neg C_2 \vee \neg C_1) \wedge \dots \wedge (\neg C_2 \vee \neg C_n) \wedge (\neg C_2 \vee P) \wedge (C_2 \vee C_1 \vee \dots \vee C_n \vee \neg P) \wedge$ $(\neg C_n \vee \neg C_1) \wedge (\neg C_n \vee \neg C_2) \wedge \dots \wedge (\neg C_n \vee \neg C_{n-1}) \wedge (\neg C_n \vee P) \wedge (C_n \vee C_1 \vee C_2$ $\vee \dots \vee C_{n-1} \vee \neg P)$

Tabla 7. Transformación de lógica proposicional a CNF, ejemplo Teléfono Móvil.

Lógica Proposicional	Transformación a CNF
$TelefonoMovil \leftrightarrow Llamadas$	$(\neg TelefonoMovil \vee Llamadas) \wedge (TelefonoMovil \vee \neg Llamadas)$
$TelefonoMovil \leftrightarrow Pantalla$	$(\neg TelefonoMovil \vee Pantalla) \wedge (TelefonoMovil \vee \neg Pantalla)$
$GPS \rightarrow TelefonoMovil$	$\neg GPS \vee TelefonoMovil$
$Media \rightarrow TelefonoMovil$	$\neg Media \vee TelefonoMovil$
$Media \leftrightarrow (Camara \vee MP3)$	$(\neg Media \vee Camara \vee MP3) \wedge (Media \vee \neg Camara) \wedge (Media \vee \neg MP3)$
$(Basica \leftrightarrow (\neg Color \wedge \neg AltaResolucion$ $\wedge Pantalla)) \wedge$ $(Color \leftrightarrow (\neg Basica \wedge \neg AltaResolucion$ $\wedge Pantalla)) \wedge$ $(AltaResolucion \leftrightarrow (\neg Basica \wedge \neg Color$ $\wedge Pantalla))$	$(\neg Basica \vee \neg Color) \wedge (\neg Basica \vee \neg AltaResolucion) \wedge (\neg Basica \vee$ $Pantalla) \wedge (Basica \vee Color \vee AltaResolucion \vee \neg Pantalla) \wedge$ $(\neg Color \vee \neg Basica) \wedge (\neg Color \vee \neg AltaResolucion) \wedge (\neg Color \vee$ $Pantalla) \wedge (Color \vee Basica \vee AltaResolucion \vee \neg Pantalla) \wedge$ $(\neg AltaResolucion \vee \neg Basica) \wedge (\neg AltaResolucion \vee \neg Color)$ $\wedge (\neg AltaResolucion \vee Pantalla) \wedge (AltaResolucion$ $\vee Basica \vee Color \vee \neg Pantalla)$
$Camara \rightarrow AltaResolucion$	$\neg Camara \vee AltaResolucion$
$\neg(GPS \wedge Basic)$	$\neg GPS \vee \neg Basica$

Forma Normal Conjuntiva – Formato DIMACS

El formato DIMACS es un formato estándar aceptado por los SAT *solvers* (SATLINK - Dimacs, 2020). Este formato está conformado por tres bloques: el primer bloque especifica los comentarios, los cuales deben iniciar con la letra c. El segundo bloque especifica la línea problema, la cual empieza con una letra p, seguida de las siglas cnf, el número de variables y el número de cláusulas. Por último, el tercer bloque lista cada una de las cláusulas en una línea diferente. Una cláusula está especificada por un conjunto de números, donde cada uno representa una variable.

La Tabla 8 muestra el paso inicial del proceso, en el cual se asigna un número a cada variable del modelo *Mobile Phone* y en la segunda columna de la Tabla 9 se especifica el formato DIMACS, el cual muestra la representación de cada una de las cláusulas de la fórmula CNF.

Tabla 8. Mapeo de características a identificadores numéricos.

Característica	Identificador numérico en el formato DIMACS
TelefonoMovil	0
Llamadas	1
Pantalla	2
GPS	3
Media	4
Camara	5
MP3	6
Basica	7
Color	8
AltaResolucion	9

Tabla 9. Transformación de CNF a formato Dimacs, ejemplo Teléfono Móvil.

CNF	Formato Dimacs			
	p cnf 10 22			
$(\neg \text{MobilePhone} \vee \text{Calls}) \wedge (\text{MobilePhone} \vee \neg \text{Calls})$	-0	1		
	0	-1		
$(\neg \text{MobilePhone} \vee \text{Screen}) \wedge (\text{MobilePhone} \vee \neg \text{Screen})$	-0	2		
	0	-2		
$\neg \text{GPS} \vee \text{MobilePhone}$	-3	0		
$\neg \text{Media} \vee \text{MobilePhone}$	-4	0		
$(\neg \text{Media} \vee \text{Camera} \vee \text{MP3}) \wedge (\text{Media} \vee \neg \text{Camera}) \wedge (\text{Media} \vee \neg \text{MP3})$	-4	5	6	
	4	-5		
	4	-6		
$(\neg \text{Basic} \vee \neg \text{Color}) \wedge (\neg \text{Basic} \vee \neg \text{HighResolution}) \wedge (\neg \text{Basic} \vee \text{Screen}) \wedge (\text{Basic} \vee \text{Color} \vee \text{HighResolution} \vee \neg \text{Screen}) \wedge$	-7	-8		
	-7	-9		
	-7	2		
	7	8	9	-2
$(\neg \text{Color} \vee \neg \text{Basic}) \wedge (\neg \text{Color} \vee \neg \text{HighResolution}) \wedge (\neg \text{Color} \vee \text{Screen}) \wedge (\text{Color} \vee \text{Basic} \vee \text{HighResolution} \vee \neg \text{Screen}) \wedge$	-8	-7		
	-8	-9		
	-8	2		
	8	7	9	-2
$(\neg \text{HighResolution} \vee \neg \text{Basic}) \wedge (\neg \text{HighResolution} \vee \neg \text{Color}) \wedge (\neg \text{HighResolution} \vee \text{Screen}) \wedge (\text{HighResolution} \vee \text{Basic} \vee \text{Color} \vee \neg \text{Screen})$	-9	-7		
	-9	-8		
	-9	2		
	9	7	8	-2
$\neg \text{Camera} \vee \text{HighResolution}$	-5	9		
$\neg \text{GPS} \vee \neg \text{Basic}$	-3	-7		

1.6. ESCALABILIDAD

La escalabilidad en lenguajes de variabilidad hace referencia a los mecanismos que proveen estos lenguajes para soportar el modelado de sistemas con un gran número de elementos (Eichelberger & Schmid, 2015).

El modelado de variabilidad a gran escala se estructura con las siguientes características (Eichelberger & Schmid, 2015):

- **Composición:** ¿Pueden los elementos configurables ser compuestos en un sólo modelo? (Czarnecki et al., 2004). Esta característica hace referencia al proceso en el que se combinan diferentes modelos de variabilidad para formar un único modelo. Esto permite especificar diferentes subsistemas y dominios de aplicación de la línea de productos en diferentes modelos de variabilidad (El-Sharkawy et al., 2011). En el caso del modelo del Teléfono Móvil (Ilustración 1), la composición permitiría unir modelos que especifican diferentes vistas del producto (i.e modelo Procesador, modelo Hardware) y generar el modelo que representa la línea de productos de teléfonos móviles.
- **Modularidad:** ¿En qué nivel de granularidad, si es que existe, se admite el modularidad? (Schmid, 2010). Esta característica hace referencia a los mecanismos que permiten descomponer los elementos dentro del modelo para ser tratados como entidades independientes, lo que permite dar mayor claridad y flexibilidad al modelo. En el caso del modelo del Teléfono Móvil (Ilustración 1), la modularidad permitiría especificar diferentes vistas del producto (i.e modelo Procesador, modelo Hardware) en unidades independientes como los modelos fraccionados.
- **Evolución:** ¿El lenguaje de variabilidad proporciona los conceptos específicos para apoyar la evolución? (Schmid, 2010). Esta característica hace referencia a los mecanismos que permiten el manejo sistemático de los cambios hechos sobre los modelos de variabilidad, sus elementos y relaciones. Permite hacer gestión sobre el versionamiento y las restricciones que tengan que ver con la forma en que interactúan estas versiones entre sí.

1.7. HIGH-LEVEL VARIABILITY LANGUAGE (HLVL)

HLVL es un lenguaje textual para el modelado de variabilidad, que busca soportar los conceptos propios de una gran variedad de lenguajes para el modelado. Su objetivo es mitigar los problemas del compartir y reusar modelos dentro de la comunidad de las SPL, por tanto, es una propuesta de lenguaje estándar para modelar variabilidad (Villota et al., 2019).

Un modelo de variabilidad o script dentro de HLVL, se compone de tres bloques principales en el siguiente orden: i) un identificador para el modelo; ii) un bloque de elementos (bloque x, Ilustración 8), donde se van a especificar los elementos que son susceptibles de variar dentro del modelo; iii) un bloque de relaciones (bloque y, Ilustración 8), donde se van a especificar las relaciones de variabilidad entre los elementos variables del modelo. En la Ilustración 8 se ejemplifica un script con la sintaxis propia de HLVL (Villota et al., 2019).

```
model TiendaVirtual

    bloque x
    elements:
        boolean tipoDeConexion,conexionSegura,conexionInsegura,pago,PayPal
        symbolic tipoDeComprador variants:['esporadico','regular']
        comment: {"Representa el tipo de comprador"}

    bloque y
    relations:
        exp1: expression(3 <= pagoTarjeta.confidencialidad AND
                        pagoTarjeta.confidencialidad <= 5)
        m1: mutex(tarjetaCredito,conexionInsegura)
        m2: mutex(tipoComprador='esporadico',[tarjetaRegalo,tarjetaCredito])
        imp1: implies(payPal,conexionSegura)
        imp2: implies(tipoComprador='regular',
                    [conexionSegura,cuentaClienteFrecuente])
        dc1: decomposition(tarjeta,[tarjetaRegalo,tarjetaDebito])[0,1]
        dc2: decomposition(tarjeta,[tarjetaCredito])[1,1]
        dc3: decomposition(implementacion,[servidorAplicacion])[1,5]
        g1: group(pago,[payPal,tarjeta])[1,*]
```

Ilustración 4. Ejemplo de un script en HLVL

1.7.1. Elementos y Variantes

Los elementos son la unidad de variabilidad en HLVL, deben ser tipados y opcionalmente pueden incluir un comentario para agregar claridad. Los tipos de elementos soportados por HLVL son booleanos, enteros y símbolos. Cada elemento está asociado a un conjunto de variantes, las cuales representan las opciones o formas en que varía el elemento (Villota et al., 2019).

1.7.2. Relaciones de Variabilidad

- **Inclusión/Exclusión:** HLVL soporta la especificación de inclusión y exclusión, y además pueden ser representadas por medio de expresiones de restricción. En la Ilustración 8 se pueden evidenciar relaciones de exclusión (m1 y m2), y relaciones de inclusión (imp1 e imp2) (Villota et al., 2019).
- **Expresiones de Restricción:** Este constructo del lenguaje tiene como objetivo incluir reglas complejas entre elementos del modelo por medio de operadores lógicos, relacionales, aritméticos y globales. Estas expresiones se especifican en el lenguaje de HLVL para expresiones, en la relación exp1 de la Ilustración 8 se evidencia un ejemplo (Villota et al., 2019).
- **Descomposición:** HLVL soporta constructos de tipo jerárquico para describir relaciones padre-hijo uno-a-uno. Para este tipo de relaciones tiene que especificar una cardinalidad, la cual especifica el número de instancias del elemento hijo que pueden estar presentes en la configuración. En la Ilustración 8 se pueden evidenciar algunos ejemplos (dc1, dc2 y dc3) (Villota et al., 2019).
- **Grupos:** HLVL soporta constructos de tipo jerárquico para describir relaciones padre-hijo uno-a-muchos. Para este tipo de relaciones tiene que especificar una cardinalidad, la cual especifica el número mínimo y máximo de hijos que pueden estar presentes en la configuración. La relación g1 en la Ilustración 8, es un ejemplo de relación de grupo (Villota et al., 2019).

2. Estado del Arte

El objetivo de este trabajo de grado es contribuir a la escalabilidad de HLVL, es decir, al soporte de al menos una de las características de composición, modularización o evolución. Por lo tanto, se tomaron en cuenta las formas en que otros lenguajes de variabilidad textuales han implementado mecanismos para soportar el modelado y análisis de modelos de variabilidad fraccionados. A continuación, se describe la forma en la que estos lenguajes implementan mecanismos que soportan las características de escalabilidad descritas en el marco teórico. El análisis comparativo descrito en esta sección está basado en la revisión sistemática de literatura sobre lenguajes textuales de (Eichelberger & Schmid, 2015), que luego fue extendido por (Ter Beek et al., 2019). Estos estudios sistemáticos de literatura presentan un esquema de clasificación de lenguajes de variabilidad que se ha convertido en el referente para evaluarlos en la comunidad de SPLE.

2.1. LENGUAJES

En esta sección se definen para ser comparados cinco de los once lenguajes presentados en la revisión sistemática de *Eichelberger y Schmid*. Los lenguajes fueron escogidos por soportar mínimo una de las características de escalabilidad.

- **VSL** es un lenguaje implementado en el *framework* Manejo de Variabilidad por Composición (CVM por sus siglas en inglés) (Abele et al., 2010). Los mecanismos de composición que proporciona son los enlaces de configuración y las entidades variables. En cuanto al soporte de modularidad provee la especificación y configuración de modelos.
- **FAMILIAR** es un lenguaje textual específico de dominios definido por Acher et al. (2013). Este lenguaje proporciona dos operadores de composición: *Merge* para modelos superpuestos y *Agregación* para modelos disjuntos. FAMILIAR permite definir los módulos a nivel de *script* y ocultar o exportar los elementos configurables.

- **TVL** es un lenguaje textual para modelar variabilidad definido por Classen et al. (2011). Posteriormente fue extendido en (Clarke et al., 2011) con el nombre **μTVL** e incorporado al *framework* para la Especificación de Comportamiento Abstracto (ABS por sus siglas en inglés). Ambos lenguajes soportan únicamente la composición, TVL por medio de la inclusión de modelos y μTVL mediante la intersección y conjunción de modelos.

- **VELVET** es un lenguaje definido por Rosenmüller et al. (2011) el cual permite describir las dimensiones de variabilidad de una PSL en modelos de variabilidad separados. Provee tres mecanismos para la composición de modelos: herencia, superposición y agregación.

- **IVML** es un lenguaje presentado por Schmid et al. (2015) el cuál soporta la composición por medio de importación de modelos, la evolución por medio de restricciones de versión sobre las importaciones y la modularidad mediante el uso de interfaces.

2.2. ANÁLISIS COMPARATIVO

Se realizó una comparativa de las propuestas en cuanto a su capacidad para dar solución al problema. En la Tabla 10 se resume la comparación entre los diferentes lenguajes propuestos como alternativas de solución.

Tabla 10. Comparación de alternativas de solución para el problema

	Composición	Evolución	Modularidad	Paradigmas de Modelado
VSL	Entidades variables	-	Modelo, configuración	Modelos de Características
FAMILIAR	<i>Merge</i> , agregación	-	-	
TVL	Inclusión	-	-	
μTVL	Conjunción/Intersección	-	-	
Clafer	Herencia	-	-	
VELVET	Herencia, <i>Superimposition</i> , Agregación	-	Superimposición	
IVML	Importación	Versionamiento de Modelos	Interfaces	Modelos de Características, Puntos de Variación, Modelos de Decisión
HLVL + Extensión	Herencia Agregación <i>Merge</i>	-	<i>Script</i>	

De la tabla comparativa es posible concluir lo siguiente:

FAMILIAR, TVL, μTVL y Clafer son lenguajes que únicamente soportan la composición de modelos; sin embargo, para el problema que plantea el proyecto, la modularidad es un

componente fundamental para la solución, ya que se requiere modelar diferentes preocupaciones de los *stakeholders* por separado.

VSL, VELVET y HLVL (junto con la extensión propuesta) atacan las características de escalabilidad que tienen un impacto significativo sobre la solución del problema planteado: modularidad y composición. Sin embargo, HLVL presenta una ventaja ya que además de abarcar la expresividad del modelado por características, como lo hacen VSL y VELVET, incluye los paradigmas de modelado orientado a puntos de variación y decisión.

La propuesta para el mejoramiento de la escalabilidad de HLVL consiste en darle mecanismos que le permitan soportar la composición y modularización de modelos. En cuanto a la composición, se propone implementar la composición a través de *Composición a través de Herencia*, soportada por operaciones para combinar modelos tales como la composición a través *En* esta sección se presentan las operaciones de análisis, las cuales definen formas para soportar las características de composición, a través de la transformación de los modelos. La composición hace referencia a el proceso en que se combinan diferentes modelos de variabilidad para formar un único modelo (Czarnecki et al., 2004).

Composición a través de *merge* y *Composición a través de agregación*.

CAPÍTULO 3: Escenarios para el Soporte de Escalabilidad de HLVL

Con el objetivo de seleccionar los escenarios de implementación para el soporte de escalabilidad de HLVL, se realizó una caracterización de las operaciones y mecanismos que soportan la modularización y composición de modelos en lenguajes de variabilidad.

La implementación de mejoras para el soporte de escalabilidad a través de estos escenarios requiere el desarrollo de un componente de modelado y otro de análisis, ya que los lenguajes necesitan de ambos para su construcción. El componente de modelado se refiere a las construcciones propias de la sintaxis del lenguaje, es decir, palabras o símbolos con significado dentro del lenguaje. El componente de análisis se refiere a las operaciones necesarias para transformar modelos. Debido a que se requiere el desarrollo de ambos componentes, se separaron las implementaciones en dos categorías: mecanismos de modelado y operaciones de análisis, en la primera se define la sintaxis en que un lenguaje especifica el mecanismo y en la segunda se definen operaciones para la composición de modelos, en las cuáles se especifica la lógica correspondiente a una sintaxis.

En cada uno de los mecanismos de modelado se presentan escenarios que proponen una sintaxis correspondiente en HLVL. Un escenario es un conjunto de mecanismos o técnicas que soportan las características de escalabilidad, agrupadas de acuerdo con los criterios definidos en la sección *Criterios de Agrupación*.

Con el objetivo de explicar cada una de las propuestas de mecanismos de modelado, se presenta un ejemplo original. Con este ejemplo se desea modelar un computador por medio de dos modelos fraccionados. El primer modelo fraccionado es la representación de un Procesador (Ilustración 9), el cual cuenta con características para la frecuencia, número de núcleos e hilos y el segundo modelo fraccionado, representa un Sistema Operativo (Ilustración 10) con características para tres tipos de sistemas operativos

distintos. Para los mecanismos de composición, se utilizan los modelos Procesador y Sistema Operativo para componer un modelo PC, el cual cuenta con características para una tarjeta gráfica. Por otro lado, para los mecanismos de modularización se utilizan los modelos de forma independiente.

```
1. model Procesador
2. elements:
3.   boolean procesador
4.   boolean frecuencia, 1GHz, 2GHz, 4GHz
5.   boolean nucleos, N2, N3, N4
6.   boolean hilos, H2, H3
7. relations:
8.   com1: common(procesador)
9.   grp1: group(frecuencia, [frecuencia, nucleos, hilos])[3,3]
10.  grp1: group(frecuencia, [1GHz, 2GHz, 4GHz])[1,1]
11.  dec2: group(hilos, [H2, H3])[1,1]
12.  grp2: group(nucleos, [N2, N3, N4])[1,*]
13.  m1:   mutex(frecuencia.1GHz, H3)
14.  imp1: implies(N4,4GHz)
```

Ilustración 5. Sintaxis en HLVL del modelo Procesador.

```
1. model SistemaOperativo
2. elements:
3.   boolean sistemaOperativo, windows, linux, macOS
4. relations:
5.   com1: common(sistemaOperativo)
6.   grp1: group(sistemaOperativo, [windows, linux, macOS])[1,1]
```

Ilustración 6. Sintaxis en HLVL para el modelo Sistema Operativo.

1. Mecanismos de Modelado

En esta sección se presentan los mecanismos de modelado, definen formas para soportar las características de composición o modularización, a través de la definición de una sintaxis para el lenguaje. La composición hace referencia a el proceso en que se combinan diferentes modelos de variabilidad para formar un único modelo (Czarnecki et al., 2004). La modularización hace referencia a los mecanismos que permiten descomponer los elementos dentro del modelo para ser tratados como entidades independientes (Schmid, 2010).

1.1. COMPOSICIÓN A TRAVÉS DE HERENCIA

La composición a través de Herencia permite crear un modelo de variabilidad que extiende de otro agregando nuevos elementos y relaciones. Existen dos casos de composición según las características de los modelos a componer, si ambos modelos poseen características equivalentes se consideran modelos superpuestos, en caso contrario, los modelos se consideran independientes. Para cada caso, se define un enfoque diferente de implementación:

- i. Para modelos superpuestos, la composición utiliza la operación de *Merge* definida por Acher (2010) o el enfoque basado en la forma lógica formal de los modelos, el cual transforma los modelos a su forma lógica, los une y posteriormente transforma el resultado a un nuevo modelo (Czarnecki & Wasowski, 2008).
- ii. Para modelos independientes, se utiliza la operación de Agregación definida por Acher (2010).

La propuesta de sintaxis definida para realizar composición a través de Herencia en HLVL es el uso de la palabra reservada *extends*, dada su simplicidad y su similitud a otros lenguajes. En la *Ilustración 7* se muestra un ejemplo de la composición del modelo PC con los modelos Procesador y Sistema Operativo, es decir, ambos modelos heredan sus características al modelo PC para que pueda definir restricciones de variabilidad. Como se puede observar es posible acceder a los elementos de los modelos heredados para especificar relaciones y restricciones de variabilidad, este acceso se hace por medio del identificador del modelo seguido de un punto.

```

1. model PC extends Procesador, SistemaOperativo
2. elements:
3.     boolean pc
4.     boolean tarjetaGrafica, msi, asus, evga
5. relations:
6.     com1: common(pc)
7.     grp1: group(pc, [Procesador.procesador,
8.                     SistemaOperativo.sistemaOperativo], [2,2])
9.     grp2: group(tarjetaGrafica, [msi, asus, evga], [1,1])
10.    dec1: decomposition(pc, [tarjetaGrafica], [0,1])
11.    mut1: mutex(msi, Procesador.N2)
12.    imp1: implies(asus, SistemaOperativo.windows)

```

Ilustración 7. Propuesta de sintaxis en HLVL de la composición a través de Herencia.

1.2. COMPOSICIÓN A TRAVÉS DE AGREGACIÓN

La composición a través de Agregación permite la unión de una instancia de SPL, es decir, una configuración del modelo, mediante su especificación dentro de otro modelo de variabilidad (Rosenmüller et al., 2011a). El objetivo de este mecanismo es simplificar la composición al permitir componer un modelo, que representa cientos o miles de configuraciones y una instancia, que representa una configuración.

La propuesta de sintaxis definida para realizar composición a través de Agregación en HLVL se compone de:

- i. Definición de un elemento cuyo tipo sea un modelo de una SPL (e.i relaciones entre clases Java) (Ilustración 8, líneas 5 y 6). Esto permite el acceso a los elementos del modelo referenciado.
- ii. Un bloque `configurations` en el cual se definen configuraciones de los modelos referenciados (Ilustración 8, línea 12).
- iii. Configuración por medio del identificador del modelo, la palabra reservada `with` y los elementos concretos que constituyen la configuración del modelo entre corchetes (Ilustración 8, líneas 13 a 16).

```

1. model PC
2. elements:
3.   boolean pc
4.   boolean tarjetaGrafica, msi, asus, evga
5.   Procesador procesador
6.   SistemaOperativo os
7. relations:
8.   com1: common(pc)
9.   grp1: group(pc, [procesador, os])[2,2]
10.  grp2: group(tarjetaGrafica, [msi, asus, evga])[1,1]
11.  dec1: decomposition(pc, [tarjetaGrafica])[0,1]
12. configurations:
13.   procesador with [procesador.nucleos.N4,
14.                   procesador.hilos.H2,
15.                   procesador.frecuencia.4GHz]
16.   os with [os.macOS]

```

Ilustración 8. Propuesta de sintaxis en HLVL de la composición a través de agregación.

1.3. COMPOSICIÓN A TRAVÉS DE *INCLUDE*

La composición a través de *Include* permite incluir dentro del modelo los contenidos de un archivo referenciado. La palabra reservada *include* es una directiva de preprocesamiento que toma como parámetro la ruta del archivo y es reemplazada por el contenido del mismo (Classen et al., 2011).

La sintaxis definida para realizar composición a través de *Include* en HLVL se compone de la palabra reservada *include* (Ilustración 10, línea 3) la cual recibe cómo parámetro la ruta a un archivo que contiene un fragmento del modelo (Ilustración 9). La Ilustración 9; **Error!** **No se encuentra el origen de la referencia.** muestra el modelo resultante.

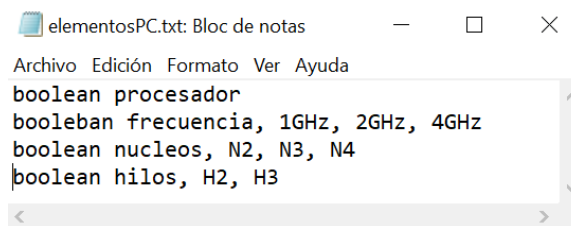


Ilustración 9. Archivo que contiene únicamente los elementos del modelo PC

```

1. model Procesador
2. elements:
3.   include("elementosPC.txt")
4. relations:
5.   com1: common(procesador)
6.   grp1: group(procesador, [frecuencia, nucleos, hilos])[3,3]
7.   grp1: group(frecuencia, [1GHz, 2GHz, 4GHz])[1,1]
8.   dec2: group(hilos, [H2, H3])[1,1]
9.   grp2: group(nucleos, [N2, N3, N4])[1,*]
10.  m1: mutex(frecuencia.1GHz, H3)
11.  imp1: implies(N4,4GHz)

```

Ilustración 10. Propuesta de sintaxis en HLVL de la composición a través de include.

Cabe resaltar que es necesario el uso de una convención de nombres para los componentes dentro del modelo para evitar problemas semánticos y de colisión.

1.4. COMPOSICIÓN A TRAVÉS DE *IMPORT*

El uso de la palabra reservada `import` indica el uso de un proyecto, es decir, permite usar ciertos elementos de un proyecto por referencia. Se puede combinar con la palabra reservada `with` para limitar la información según una versión del proyecto (Schmid et al., 2015).

La sintaxis definida para realizar composición a través de *Import* en HLVL se conforma de:

- i. La palabra reservada `import`, la cual permite hacer referencia a un modelo externo y sus componentes (Ilustración 11, líneas 1 y 2).
- ii. La palabra reservada `as`, la cual permite asignar un identificador al componente importado para ser usado dentro del modelo (Ilustración 11, líneas 1 y 2).

```

1. import Procesador.elementos.procesador as proc
2. import SistemaOperativo.elementos.sistemaOperativo as os
3. model PC
4. elements:
5.   boolean pc
6.   boolean tarjetaGrafica, msi, asus, evga
7. relations:
8.   com1: common(pc)
9.   grp1: group(pc, [proc, os])[2,2]
10.  grp2: group(tarjetaGrafica, [msi, asus, evga])[1,1]
11.  dec1: decomposition(pc, [tarjetaGrafica])[0,1]
12.  imp1: implies(tarjetaGrafica.msi, os.windows)

```

Ilustración 11. Propuesta de sintaxis en HLVL de la composición a través de import.

1.5. COMPOSICIÓN Y MODULARIZACIÓN TRAVÉS DE SUPERPOSICIÓN

El mecanismo de Superposición permite descomponer el modelo en múltiples dimensiones o aspectos sin necesidad de generar nuevos modelos de variabilidad, es decir, definir divisiones explícitas dentro del mismo modelo (Rosenmüller et al., 2011a).

La sintaxis definida para realizar composición y modularización a través de Superposición en HLVL (Ilustración 16) consiste en el uso de la palabra reservada `submodel`. Los submodelos son declarados al mismo nivel que el modelo principal y pueden acceder a cualquier elemento definido en el archivo.

```

1. model PC
2. elements:
3.   boolean pc
4.   boolean tarjetaGrafica, msi, asus, evga
5. relations:
6.   com1: common(pc)
7.   grp1: group(pc, [procesador])[1,1]
8.   grp2: group(tarjetaGrafica, [msi, asus, evga])[1,1]
9.   dec1: decomposition(pc, [tarjetaGrafica])[0,1]
10.  imp1: implies(tarjetaGrafica.msi, os.windows)
11.
12. submodel Procesador
13. elements:
14.   boolean procesador
15.   boolean frecuencia, 1GHz, 2GHz, 4GHz
16.   boolean nucleos, N2, N3, N4
17.   boolean hilos, H2, H3
18. relations:
19.   com1: common(procesador)
20.   grp1: group(procesador, [frecuencia, nucleos, hilos])[3,3]
21.   grp1: group(frecuencia, [1GHz, 2GHz, 4GHz])[1,1]
22.   dec2: group(hilos, [H2, H3])[1,1]
23.   grp2: group(nucleos, [N2, N3, N4])[1,*]
24.   m1: mutex(frecuencia.1GHz, H3)
25.   imp1: implies(N4,4GHz)

```

Ilustración 12. Propuesta de sintaxis en HLVL de la modularización y composición a través de Superposición.

1.6. MODULARIZACIÓN TRAVÉS DE INTERFACES

Las interfaces son un mecanismo que permite limitar la visibilidad de los elementos dentro de un modelo (Schmid et al., 2015). Una interfaz debe ser declarada mediante la palabra reservada `interface` al inicio del modelo y debe especificar mediante la palabra reservada `export` los elementos que deben ser visibles cuando la interfaz sea importada.

La sintaxis definida para realizar modularización a través de interfaces en HLVL se compone de:

- i. Un bloque `interfaces`, el cual define las interfaces del modelo (Ilustración 17, línea 2).
- ii. Declaración de una interfaz, la cual consta de un identificador y la palabra reservada `exports` seguida de los elementos que serán visible bajo la interfaz entre corchetes (Ilustración 17, líneas 3, 4 y 5).

```
1. model Procesador
2. interfaces:
3.   gamaAlta exports [4GHz, N4, H3]
4.   gamaMedia exports [2GHz, N3, H2]
5.   gamaBaja exports [1GHz, N2, H2]
6. elements:
7.   boolean procesador
8.   boolean frecuencia, 1GHz, 2GHz, 4GHz
9.   boolean nucleos, N2, N3, N4
10.  boolean hilos, H2, H3
11. relations:
12.  com1: common(procesador)
13.  grp1: group(procesador, [frecuencia, nucleos, hilos])[3,3]
14.  grp1: group(frecuencia, [1GHz, 2GHz, 4GHz])[1,1]
15.  dec2: group(hilos, [H2, H3])[1,1]
16.  grp2: group(nucleos, [N2, N3, N4])[1,*]
17.  m1: mutex(frecuencia.1GHz, H3)
18.  imp1: implies(N4,4GHz)
```

Ilustración 13. Propuesta de sintaxis en HLVL de la modularización a través de interfaces

Para hacer uso de la interfaz de un modelo basta con importarlo especificando la interfaz (Ilustración 18, línea 1).

```
1. import Procesador.interfaces.gamaMedia as procGamaMedia
2. model PC
3. elements:
4.   boolean pc
5.   boolean tarjetaGrafica, msi, asus, evga
6. relations:
7.   com1: common(pc)
8.   grp1: group(pc, [procGamaMedia])[1,1]
9.   grp2: group(tarjetaGrafica, [msi, asus, evga])[1,1]
10.  dec1: decomposition(pc, [tarjetaGrafica])[0,1]
11.  imp1: implies(tarjetaGrafica.msi, os.windows)
```

Ilustración 14. Propuesta de sintaxis en HLVL del uso de interfaces

2. Operaciones de Análisis

En esta sección se presentan las operaciones de análisis, las cuales definen formas para soportar las características de composición, a través de la transformación de los modelos. La composición hace referencia a el proceso en que se combinan diferentes modelos de variabilidad para formar un único modelo (Czarnecki et al., 2004).

2.1. COMPOSICIÓN A TRAVÉS DE *MERGE*

La composición a través de *Merge*, es una operación que permite fusionar modelos que comparten varias características y/o son vistas diferentes de un aspecto del sistema (Acher, 2011). Este tipo de composición se puede dar de tres modos distintos: unión, intersección y diferencia. Esta especificación de composición está limitada al uso de modelos de características.

2.1.1. Unión

La composición a través de *Merge* en modo unión, tiene como resultado un modelo de características cuyo conjunto de configuraciones válidas contiene todas las configuraciones de los modelos de características de entrada. La información original del modelo *Base* debe conservarse al agregar información del modelo *Aspect*. La composición de dos modelos (*Base* y *Aspect*) es un nuevo modelo donde cada configuración es válida en uno de los dos modelos.

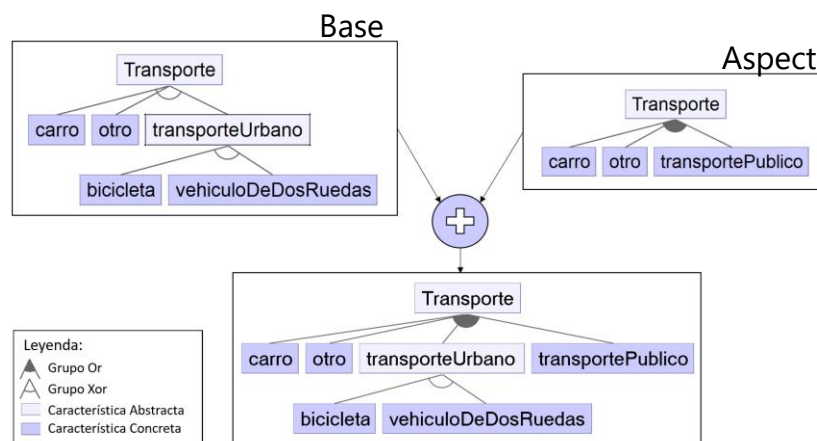


Ilustración 15. Ejemplo gráfico del funcionamiento de la operación de Merge en modo unión tomado de ((Acher, 2011)).

2.1.2. Intersección

Al realizar composición a través de *Merge* en modo intersección, sólo se conserva la información que es común en el conjunto de *feature models* de entrada. De la composición de dos modelos (*Base* y *Aspect*) se obtiene un nuevo modelo donde cada configuración es válida en ambos modelos.

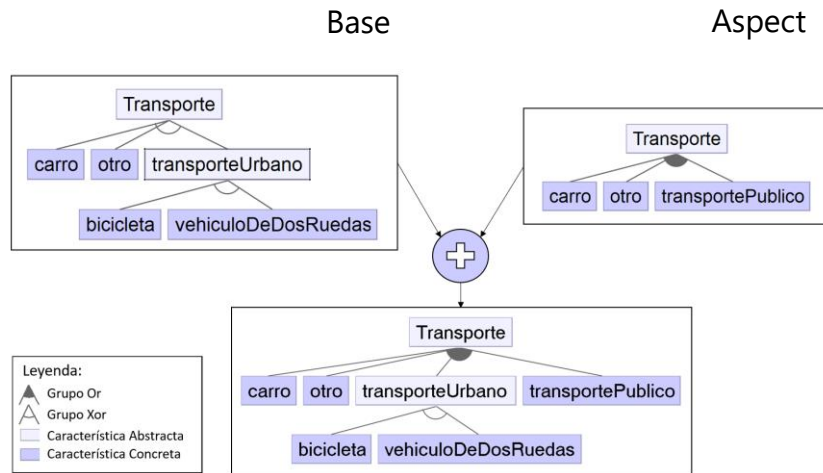


Ilustración 16. Ejemplo gráfico del funcionamiento de la operación de *Merge* en modo intersección tomado de (Acher, 2011)

2.1.3. Diferencia

Cuando se realiza la composición a través de *Merge* en modo diferencia, entre un *feature model* FM_1 y un *feature model* FM_2 , sólo se conserva la información que se encuentra en el modelo FM_1 y que no está en el modelo FM_2 . La composición de dos modelos (*Base* y *Aspect*) es un nuevo modelo donde cada configuración sólo es válida en el primero (*Base*). Cabe resaltar que el orden de entrada de los modelos de características afecta el modelo resultante.

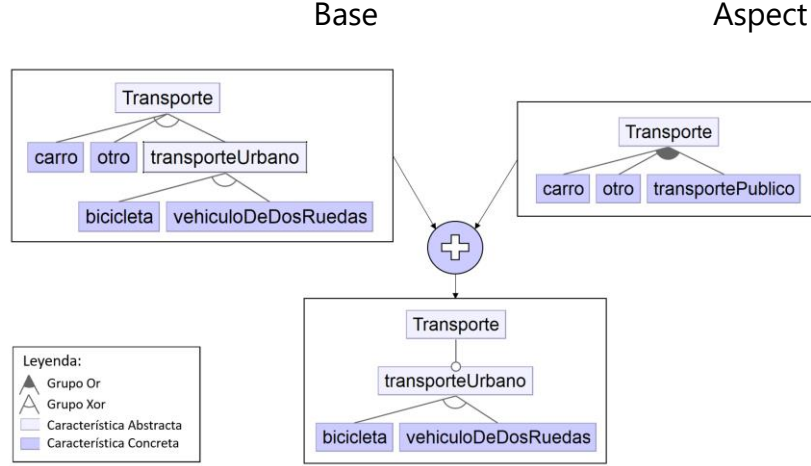


Ilustración 17. Ejemplo gráfico del funcionamiento de la operación de *Merge* en modo diferencia tomado de (Acher, 2011)

2.1.4. Enfoques de Implementación

Existen dos enfoques principales al momento de implementar la operación de *Merge*: el enfoque sintáctico y el enfoque semántico. En el enfoque sintáctico, se hace uso de la sintaxis del lenguaje para modificar la estructura de los modelos (elementos, relaciones, etc), y construir un modelo nuevo que represente la composición de ambos. En el enfoque semántico, se hace uso de la representación en lógica proposicional de cada uno de los modelos para componerlos (Acher et al., 2010a).

Inicialmente, se optó por implementar la operación de *Merge* con el enfoque sintáctico, sin embargo, en el proceso se encontraron algunas dificultades. Al tomar en cuenta la opinión de dos expertos en el tema, se tomó la decisión de cambiar de enfoque, ya que, al contrario del enfoque sintáctico, el semántico si cuenta con una definición de lógica formal presentada en (Acher, 2011). Para el enfoque semántico, Acher (2011) define la composición de modelos a través de un conjunto de fórmulas proposicionales, que varían de acuerdo al modo de *Merge* especificado. Para la construcción de estas fórmulas se utilizan las siguientes definiciones:

φ_{FM_i} : Representación CNF del i-ésimo modelo de características.

\mathcal{F}_{FM_i} : Conjunto de características del i-ésimo modelo de características.

$not(\mathcal{F}_{FM_i})$: Negación del conjunto de características del i-ésimo modelo de características.

$\mathcal{F}_{FM_a} \setminus \mathcal{F}_{FM_b}$: Conjunto de características que pertenecen al modelo de características FM_a y no al modelo de características FM_b .

Las siguientes formulas proposicionales especifican la composición de dos modelos de variabilidad FM_1 y FM_2 a través de la operación de *Merge*. La información usada de cada modelo es la 1.5.1 (φFM_i) y su conjunto de características (\mathcal{F}_{FM_i}).

Unión

En este caso la unión hace referencia a la unión estricta definida por Acher (2011), en la cual el modelo resultante tiene exactamente las configuraciones válidas de ambos modelos.

$$\varphi Resultado = (\varphi FM_1 \wedge \text{not}(\mathcal{F}_{FM_2} \setminus \mathcal{F}_{FM_1})) \vee (\varphi FM_2 \wedge \text{not}(\mathcal{F}_{FM_1} \setminus \mathcal{F}_{FM_2}))$$

Intersección

En este caso, la operación de *Merge* en modo intersección de dos modelos de variabilidad está definida por:

$$\varphi Resultado = (\varphi FM_1 \wedge \text{not}(\mathcal{F}_{FM_2} \setminus \mathcal{F}_{FM_1})) \wedge (\varphi FM_2 \wedge \text{not}(\mathcal{F}_{FM_1} \setminus \mathcal{F}_{FM_2}))$$

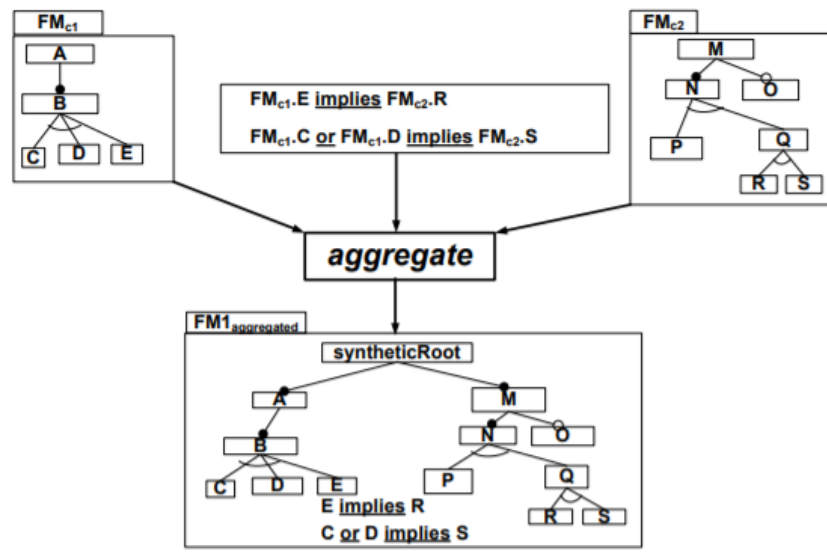
Diferencia

En este caso, la operación de *Merge* en modo diferencia de dos modelos de variabilidad está definida por:

$$\varphi Resultado = (\varphi FM_1 \wedge \text{not}(\mathcal{F}_{FM_2} \setminus \mathcal{F}_{FM_1})) \wedge \neg(\varphi FM_2 \wedge \text{not}(\mathcal{F}_{FM_1} \setminus \mathcal{F}_{FM_2}))$$

2.2. COMPOSICIÓN A TRAVÉS DE AGREGACIÓN

Esta operación tiene como objetivo relacionar modelos de variabilidad por medio de relaciones especificadas en lógica proposicional. Los modelos (de entrada) son agregados bajo una raíz llamada *synthetic*. De esta forma, las características raíz de los modelos de entrada son hijas obligatorias de la raíz *synthetic* (Acher et al., 2013). Además, se recibe como entrada un conjunto de restricciones en lógica proposicional que son agregadas al modelo resultante. En la Ilustración 18 se muestra un ejemplo básico de la operación de agregación.



(a) aggregation: basic example

Ilustración 18. Ejemplo de agregación gráfico del funcionamiento de la operación de agregación (Acher et al., 2010b).

3. Criterios de Agrupación

Con el objetivo de definir los escenarios concretos para el soporte de escalabilidad de HLVL, se analizaron los mecanismos de modelado y las operaciones de análisis. Como resultado de este análisis se obtuvieron los siguientes criterios de clasificación:

Según la cantidad de modelos:

- i. **Composición de modelos múltiples:** mecanismos que permiten la unión de modelos fraccionados completos especificados en archivos distintos.
- ii. **Composición de modelo único:** mecanismos que permiten la unión de fragmentos que hacen parte de un mismo modelo.

Según la cantidad de archivos:

- iii. **Modularización en archivos múltiples:** mecanismos que permiten la división de los componentes de un modelo en diferentes archivos.
- iv. **Modularización en archivo único:** mecanismos que permiten la división de los componentes de un modelo en un archivo.

Debido a que cada escenario corresponde a una funcionalidad concreta, estos criterios permiten identificar y agrupar las operaciones de acuerdo con su similitud o equivalencia. En la [Tabla 5](#) se muestra la clasificación de los ejemplos de sintaxis concretas y lógica de composición de acuerdo con los criterios definidos. Los colores para cada mecanismo u operación especifican los criterios a los que está asociado y ayudaron a definir los escenarios presentados en la siguiente sección. En el caso de la columna Modelos Múltiples, existen dos colores porque a pesar de que todas las operaciones y mecanismos permiten la composición de modelos múltiples, los azules lo hacen a nivel de la sintaxis del lenguaje y los naranjas a nivel de operaciones sobre los modelos.

Tabla 11. Clasificación de mecanismos de modelado y operaciones de análisis.

		Composición		Modularización	
		Modelos múltiples	Modelo único	Archivos múltiples	Archivo único
Mecanismos de modelado	Herencia	x			
	Agregación	x			
	Superposición	x			x
	<i>Import</i>	x			
	Interfaces				x
	<i>Include</i>		x		
Operaciones de análisis	Operación de <i>Merge</i>	x			
	Operación de Agregación	x			

4. Escenarios

En la Tabla 12 se especifican los cuatro escenarios resultantes de agrupar los ejemplos de sintaxis y lógica composicional de acuerdo con los cuatro criterios definidos. Cabe resaltar que dichas agrupaciones no significan que las operaciones o mecanismos sean equivalentes semánticamente.

Tabla 12. Escenarios de implementación para el soporte de escalabilidad de HLVL

Escenario	Operaciones/Mecanismos	Justificación
Composición de modelos múltiples (modelado)	Herencia Agregación Superposición <i>Import</i>	Se agruparon estos ejemplos debido a que su sintaxis hace referencia al mecanismo de unión de modelos especificados en archivos distintos.
Composición de modelos múltiples (análisis)	Operación de <i>Merge</i> Operación de Agregación	Se agruparon estos ejemplos debido a que utilizan el mecanismo de unión de modelos especificados en archivos distintos
Composición de un modelo único	<i>Include</i>	Se generó esta clasificación debido a que utiliza el mecanismo de unión de fragmentos de un modelo único.
Modularización en un archivo único	Superposición Interfaces	Se agruparon estos ejemplos debido a que su sintaxis permite la modularización de modelos en un mismo archivo

5. Niveles

Teniendo en cuenta el costo de tiempo y la curva de aprendizaje percibida para la implementación de cada una de las operaciones y mecanismos, se especificaron diferentes niveles para cada escenario (Tabla 13). Estos niveles nos permitirán definir de forma más precisa el orden en que se implementarán los escenarios.

Tabla 13. Niveles de dificultad de los escenarios según su implementación

Escenario	Nivel 1	Nivel 2	Nivel 3
Composición de modelos múltiples (modelado)	Superposición	Superposición + (Herencia o <i>Import</i>)	Superposición + (Herencia o <i>Import</i>) + Agregación
Composición de modelos múltiples (análisis)	Operación de Agregación	Operación de Agregación + Operación de Merge	-
Composición de un modelo único	<i>Include</i>	-	-
Modularización en un archivo único	Superposición	Superposición + Interfaces	-

6. Ruta de implementación

Una vez definidos los escenarios y sus respectivos niveles, es necesario trazar una ruta de implementación que indique el orden en el cual se desarrollarán los mecanismos propuestos. Para esto se realizó un diagrama (Ilustración 19) en el cual se especificaron las dependencias entre los escenarios y niveles para tener en cuenta al momento de escoger la ruta concreta.

La dependencia doble entre el escenario Composición de Modelos Múltiples (análisis) y el nivel 2 del escenario Composición de Modelos Múltiples (modelado), hace referencia a la necesidad del lenguaje de tener una sintaxis concreta y una lógica de composición asociada. Es decir, si al momento de construir un lenguaje de variabilidad se implementa alguno de los mecanismos de modelado definidos (Herencia o *Import*), es necesario la implementación de al menos una de las operaciones de análisis (Agregación o *Merge*). Así mismo, si se implementa alguna de las operaciones de análisis, se debe implementar al menos uno de los mecanismos definidos.

La dependencia entre Interfaces y el nivel 2 del escenario Composición de Modelos Múltiples (modelado), hace referencia a que el uso de las interfaces declaradas necesita un mecanismo que permita incluirlas en otros modelos.

El desarrollo de los escenarios, dada la metodología usada, se hizo de forma iterativa e incremental, de tal forma que al final de cada iteración el resultado fuera un producto funcional. Además, al inicio de cada iteración, se tuvieron en cuenta las características de los escenarios ya implementados para cumplir con las dependencias definidas en la Ilustración 19.

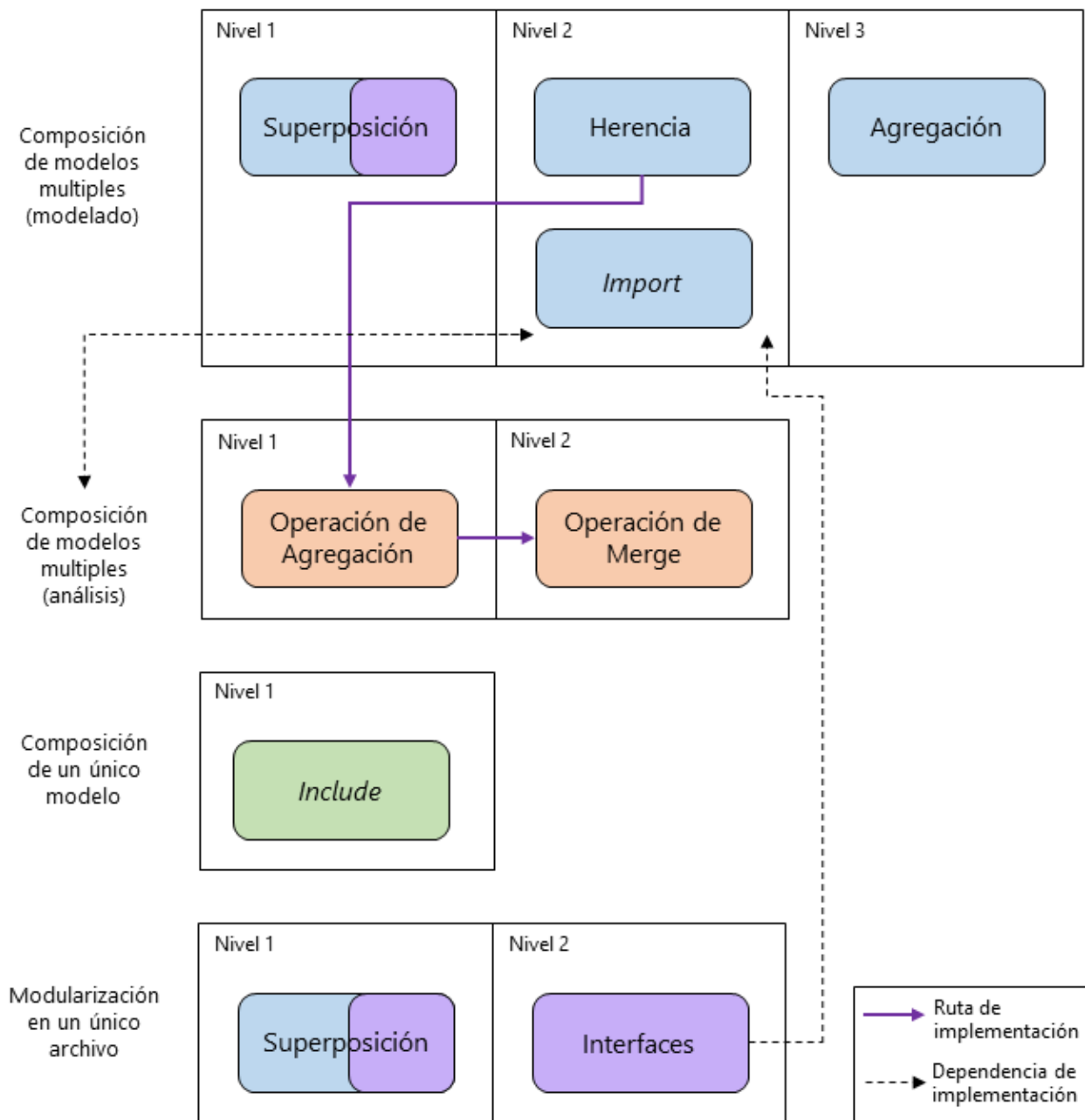


Ilustración 19. Diagrama de dependencias y ruta de implementación de escenarios.

Con el objetivo de cubrir la característica de composición, la ruta planteada abarca el mecanismo de Herencia y las operaciones de Agregación y de *Merge*. El mecanismo de Herencia cumple con la parte sintáctica de la composición de modelos especificados en distintos archivos. Dado que existe una dependencia de implementación con el escenario Composición de modelos múltiples

(análisis), la ruta abarca las operaciones de Agregación y de *Merge*. Estas operaciones permiten la composición de modelos en su totalidad (independientes y superpuestos). Lo anterior permite cumplir con los objetivos y mantenerse dentro del alcance del proyecto.

CAPÍTULO 4: Diseño de la Extensión

En este capítulo se especificará el diseño de la extensión desarrollada para HLVL. La extensión está conformada por dos componentes, uno de modelado y uno de análisis. El componente de modelado hace referencia a los mecanismos propios de la sintaxis del lenguaje, específicamente el mecanismo de Herencia. El componente de análisis hace referencia a las operaciones de análisis que permiten la composición de modelos, específicamente las operaciones de Agregación y *Merge*.

1. Herencia

La Herencia es el mecanismo de modelado que permite crear un modelo de variabilidad que extiende de otro agregando nuevos elementos y relaciones. El diseño de este mecanismo abarca: (i) la sintaxis especificada en notación *Extended Backus-Naur Form* (EBNF); (ii) la semántica, que describe el significado del mecanismo, su funcionamiento y repercusiones y (iii) los posibles cambios a la lógica de transformación de los modelos al paradigma lógico si los hubo.

La notación EBNF, está conformadas por reglas de producción de la forma $\langle \text{expr} \rangle = \langle \text{expr} \rangle$. Donde la expresión de la izquierda es un no terminal y la de la derecha es una combinación de terminales, no terminales y símbolos. Los no terminales deben ser encerrados entre antilambdas ($\langle \rangle$) y los terminales entre comillas dobles (""). Los símbolos especiales de interrogación (?), suma (+) y multiplicación (*) pueden ser especificados luego de una expresión terminal o no terminal e indican que la expresión es opcional, aparece mínimo una vez o aparece cero o más veces, respectivamente.

1.1. SINTAXIS

La declaración de un modelo que hereda de otros en HLVL se especifica de la siguiente forma:

$$\begin{aligned} \langle model \rangle = & \text{"model"} \langle name \rangle \overbrace{(\text{"extends"} \langle model-id \rangle^+)^?}^{\text{Extensión}} \\ & \text{"elements:"} \langle element \rangle^+ \\ & \text{"relations:"} \langle relation \rangle^+ \end{aligned}$$

Ilustración 20. Declaración modelos heredados, extensión HLVL

Un modelo en HLVL se compone de una declaración para el nombre del modelo, un conjunto de elementos y un conjunto de relaciones. Esta definición se extendió añadiendo una sintaxis que permite declarar modelos heredados (Ilustración 20). La parte opcional de la expresión, que se encuentra entre paréntesis, es la que especifica la declaración de modelos heredados mediante el terminal `extends` y un listado de identificadores de modelos externos.

Una vez se ha heredado de un modelo, es posible referenciar sus elementos especificando su identificador, seguido de un punto y el nombre del elemento, de la siguiente forma (Ilustración 21):

$$\langle external-element \rangle = \langle model-id \rangle \text{"."} \langle element-id \rangle$$

Ilustración 21. Referenciación de elementos de modelos heredados, extensión HLVL

1.2. SEMÁNTICA

La herencia es el mecanismo que provee HLVL para soportar la composición de modelos y permitir el modelado a gran escala, lo cual permite dar mayor flexibilidad al modelado y mejor entendimiento de modelos complejos a los diseñadores. El modelo extendido hereda los elementos del modelo del cual extiende, esto le permite referenciar elementos externos en relaciones de variabilidad de implicación y exclusión mutua. Al momento de analizar la variabilidad de un modelo que hereda de otros es necesario utilizar operaciones de composición de modelos. Dependiendo de la relación entre los elementos de los modelos, se utiliza un tipo de composición específico:

Composición de modelos superpuestos: se utiliza la operación *Merge* para componer los modelos, puesto que contienen elementos equivalentes. Existen tres modalidades de merge: union, diferencia e intersección. En los ejemplos de la sección de Composición a través de merge se puede observar el proceso para cada una de las modalidades.

Composición de modelos independientes: se utiliza la operación de Agregación para componer los modelos, debido a que no tienen elementos equivalentes. La sección de Composición a través de agregación muestra el proceso de la operación de Agregación, en el cual se tienen como entrada los modelos *c1*, *c2* y un conjunto de restricciones, cómo resultado se obtiene el modelo compuesto *aggregated* el cual mantiene las restricciones entre los modelos originales.

1.3. LÓGICA DE TRANSFORMACIÓN

La transformación de los modelos al paradigma lógico no tiene ningún cambio, puesto que, al momento de hacer análisis de un modelo fraccionado, cada uno de los modelos es transformado por separado al paradigma lógico y compuesto posteriormente.

2. Operación de Agregación

La operación de Agregación tiene como objetivo relacionar modelos de variabilidad por medio de relaciones especificadas en lógica proposicional. El resultado de esta operación es un modelo compuesto el cual contiene los modelos de entrada, unidos a través de una raíz sintética. Con este resultado el usuario es libre de especificar las respectivas restricciones de variabilidad (relaciones de implicación y/o exclusión mutua) entre los elementos del nuevo modelo.

2.1. DISEÑO

A continuación, se presenta el pseudocódigo de la operación de Agregación.

```
aggregate(models: Model[], modelName: String)
begin
    aggregatedModel := new Model()
    aggregatedModel.addHeader(modelName)
    addElements(aggregatedModel, modelName, models)
    addRelations(aggregatedModel, modelName, models)
    return aggregatedModel
end
```

Ilustración 22. Pseudocódigo operación de Agregación.

Se define la operación de Agregación, la cual recibe como parámetros el conjunto de modelos a ser compuestos y el nombre de la raíz sintética del modelo resultante. Se genera un modelo vacío y se procede a incorporar la declaración del modelo, los elementos y relaciones con el llamado a los métodos `addHeader`, `addElements` y `addRelations` respectivamente.

```
addElement(aggregatedModel: Model, modelName: String, models: Model[])
begin
    aggregatedModel.addElementBlock()
    aggregatedModel.addRootElement(modelName)
    foreach model  $\in$  models do
        foreach element  $\in$  model.getElements() do
            aggregatedModel.addElement(element)
        od
    od
end
```

Ilustración 23. Pseudocódigo método addElements.

El método `addElement` recibe como parámetros el modelo agregado en construcción, el nombre del modelo y los modelos de entrada. Su objetivo es incorporar los elementos de los modelos a ser agregados al modelo resultante. Primero añade la etiqueta *"elements:"* que representa el inicio del bloque de elementos. Después añade el elemento raíz del modelo. Por último, se agregan los elementos de cada uno de los modelos de entrada.

```

addRelations(aggregatedModel: Model, modelName: String, models: Model[])
begin
    aggregatedModel.addRelationsBlock()
    commonIds := new List()
    commonIds.add(modelName)
    forEach model ∈ models do
        forEach relation ∈ model.getRelations() do
            if ¬isCommonRelation(relation) then
                aggregatedModel.addRelation(relation)
            else
                commonIds.addAll(relation.getElements())
            fi
        od
    od
    aggregatedModel.addCommonRelation(commonIds)
end

```

Ilustración 24. Pseudocódigo método addRelations.

El método addRelations recibe como parámetros el modelo agregado en construcción, el nombre del modelo y los modelos de entrada. Su objetivo es incorporar las relaciones de los modelos a ser agregados al modelo resultante. Primero añade la etiqueta "*relations:*" que indica el inicio del bloque de relaciones. Después, inicializa una lista de identificadores de los elementos que pertenecen a una relación de tipo *common* y añade la raíz del nuevo modelo. Esta lista tiene como objetivo generar una nueva relación de *common* para el modelo resultante que agrupe todos los elementos en relaciones *common* de los modelos de entrada. Por último, se agregan las relaciones de cada uno de los modelos y la única relación *common* con la lista commonIds.

2.2. IMPLEMENTACIÓN

A continuación, se presenta el diagrama de secuencia de la operación de Agregación según la implementación para HLVL.

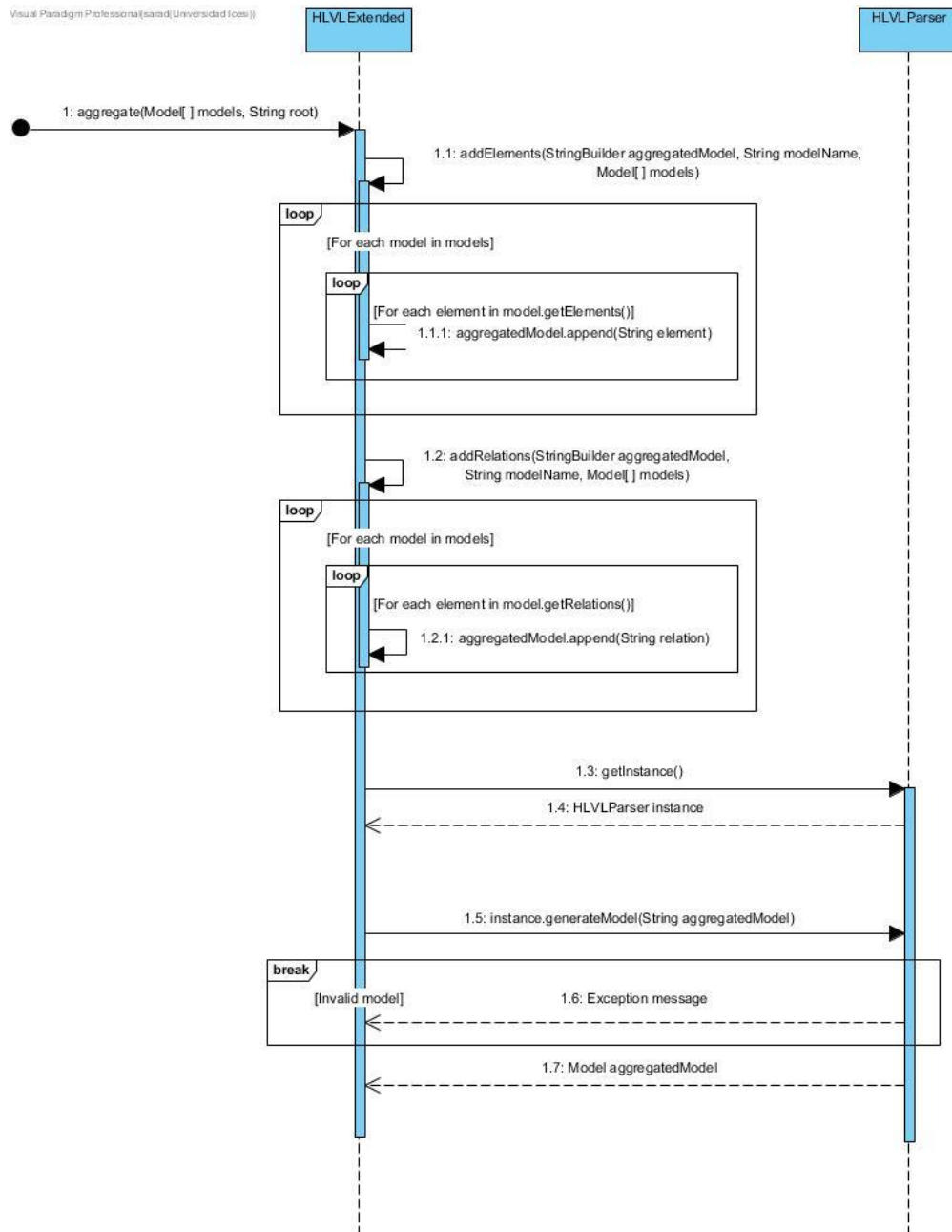


Ilustración 25. Diagrama de secuencia de la operación de Agregación.

La ejecución inicia con la invocación al método principal `aggregate` el cual toma como parámetros el conjunto de modelos a ser compuestos y el nombre de la nueva raíz para el modelo agregado. Dentro de este método se realizan tres funciones principales: `addElements`, `addRelations` y `generateModel`. El método `addElements` recibe como

parámetros la cadena equivalente al modelo que se está construyendo, su nombre, y los modelos de entrada. Aquí se procederá a incluir los elementos de los modelos de entrada a la cadena. El método `addRelations` recibe los mismos tres parámetros, sin embargo, su objetivo es incluir las relaciones de variabilidad de los modelos de entrada a la cadena. El método `generateModel` recibe como parámetro la cadena que representa el modelo agregado y retorna una estructura de datos que posteriormente permite pasar a la etapa de análisis. Para poder invocar el método `generateModel` es necesario obtener una instancia de la clase `HLVLParse` a través del método `getInstance`.

La implementación de esta propuesta puede encontrarse en el siguiente repositorio:

<https://github.com/coffeeframework/ModelOperations/blob/master/src/main/java/com/github/coffeeframework/hlvExtended/Aggregation.java>

3. Operación de Merge

La operación de *Merge* tiene como objetivo unir dos modelos mediante la fusión de sus características equivalentes, es decir, características que tienen el mismo nombre en ambos modelos. Esto permite la especificación de distintos aspectos o dominios de aplicación que pertenecen a la misma línea de productos, usando modelos independientes (Acher, 2011).

3.1. DISEÑO

A continuación, se presenta el pseudocódigo de la operación de *Merge*.

```
merge(models: Model[], mode: MergeMode)
begin
    result := CNF(models[0])
    for i = 1 to models.size do
        cnfA := CNF(models[i])
        cnfB := result
        featuresA := cnfA.getFeatures()
        featuresB := cnfB.getFeatures()
        case mode of
            UNION      : result := (cnfA  $\wedge$  not(featuresB - featuresA))  $\vee$  (cnfB  $\wedge$  not(featuresA - featuresB))
            INTERSECTION: result := (cnfA  $\wedge$  not(featuresB - featuresA))  $\wedge$  (cnfB  $\wedge$  not(featuresA - featuresB))
            DIFFERENCE  : result := (cnfA  $\wedge$  not(featuresB - featuresA))  $\wedge$   $\neg$ (cnfB  $\wedge$  not(featuresA - featuresB))
        endcase
    od
    return result
end
```

Ilustración 26. Pseudocódigo operación de Merge.

Se define la operación de *merge*, la cual recibe como parámetros el conjunto de modelos a ser compuestos y la modalidad del *merge*. Primero, se inicializa la variable *result* con la representación CNF del primer modelo, esta variable contendrá el resultado parcial y final de la composición. Después, por cada una de las iteraciones sobre los modelos restantes, se realiza: (i) se genera la representación CFN del modelo actual y se asigna a la variable *cnfA*, (ii) se asigna la variable *result* a la variable *cnfB*, (iii) se obtienen los *features* de *cnfA* y *cnfB* y se asignan a *featuresA* y *featuresB* respectivamente y (iv)

dependiendo de la modalidad del *merge* se computa la fórmula (2.1.42.1.4) y se asigna a la variable *result*. Por último, se retorna la variable *result* con el resultado final de la composición.

3.2. IMPLEMENTACIÓN

A continuación, se presenta el diagrama de secuencia de la operación de *Merge* según la implementación para HLVL.

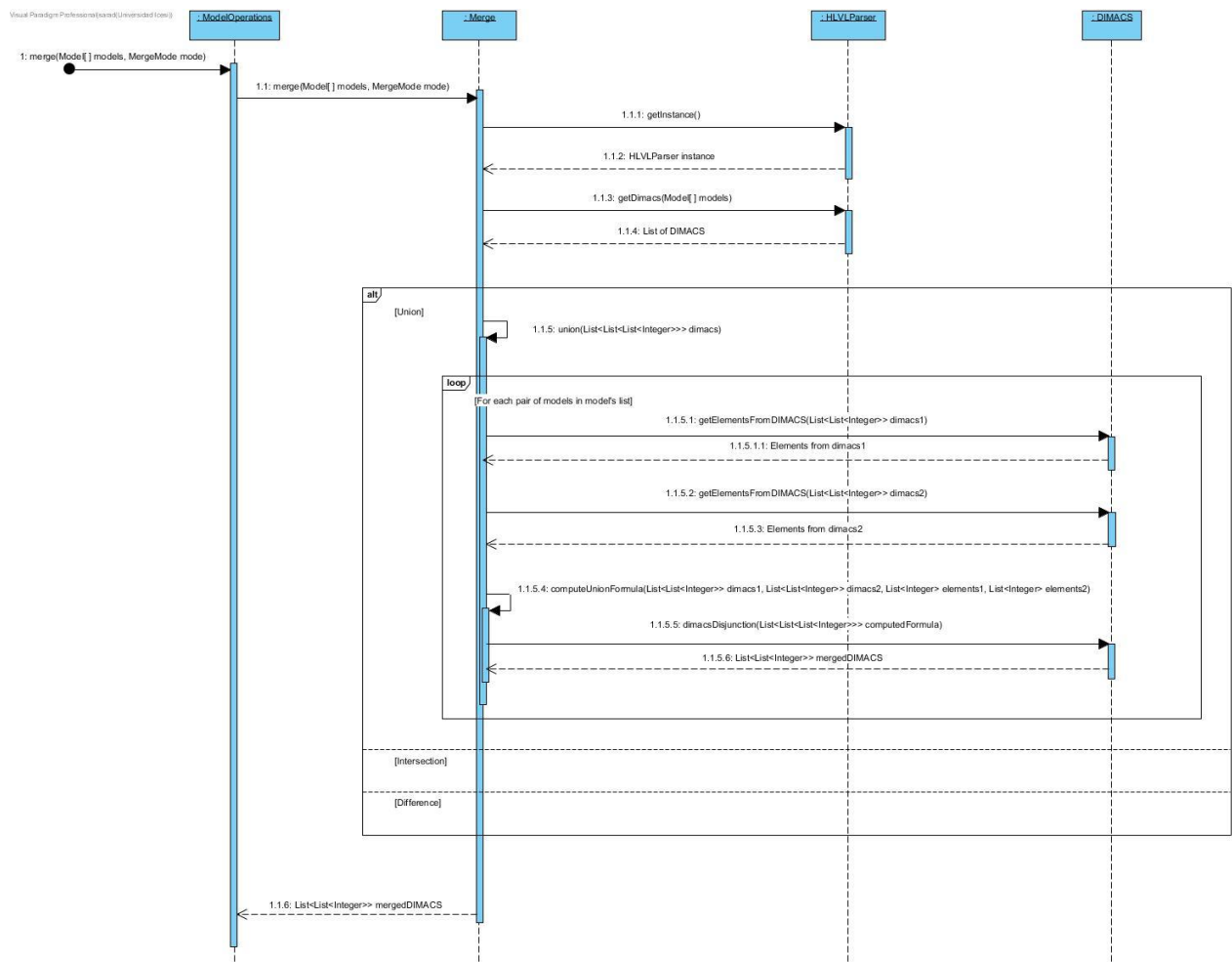


Ilustración 27. Diagrama de secuencia de la operación de *Merge*.

La ejecución inicia con la invocación al método principal *merge* el cual toma como parámetros el conjunto de modelos a ser compuestos y el modo con el cual se realizará

el *Merge*. Esta operación hace uso del formato DIMACS de los modelos para poder manipularlos en su Forma Normal Conjuntiva – **Formato DIMACS 0** y obtener las fórmulas proposicionales descritas en la sección de Composición a través de *merge0*. La versión actual de la operación de Merge sólo cuenta con la implementación del modo Unión. Sin embargo, se codificó la estructura de llamados dentro del método para que el trabajo a futuro sólo implique el desarrollo de los dos modos restantes. Para poder obtener el formato DIMACS de los modelos, es necesario invocar el método *getDimacs* a través de una instancia de la clase *HLVLParse*r. Una vez se obtienen los DIMACS de los modelos, se comienza el proceso para computar la fórmula proposicional que representa la unión. Por cada par de modelos se realizan las siguientes operaciones: Primero, se obtienen los elementos de cada modelo a través de su formato DIMACS. Segundo, se invoca al método *computeUnionFormula*, el cuál toma como parámetros los DIMACS y los elementos del par de modelos para construir y retornar los dos términos principales de la fórmula proposicional de la unión. Tercero, se invoca el método *dimacsDisjunction*, el cual toma como parámetros los dos términos resultantes del segundo paso y realiza una disyunción entre ellos para completar la fórmula proposicional. El resultado del tercer paso se acumula para ser usado en la siguiente iteración y así sucesivamente formar la unión de los modelos.

La implementación de esta propuesta puede encontrarse en el siguiente repositorio: <https://github.com/coffeeframework/ModelOperations/blob/master/src/main/java/com/github/coffeeframework/hlviExtended/Merge.java>

CAPÍTULO 5: Evaluación de la Extensión

1. Contexto

Con el objetivo de evaluar los componentes de modelado y análisis de la extensión desarrollada durante el proyecto, se llevaron a cabo dos estrategias de evaluación: (i) una encuesta a expertos que evaluó la propuesta del mecanismo de Herencia y (ii) una evaluación por casos de estudio, que evalúa las operaciones de Operación de Agregación y Operación de Merge. En este capítulo se presentan la planeación, análisis, resultados y conclusiones de las evaluaciones de cada uno de los componentes.

2. Componente de modelado

El componente de modelado de la extensión hace alusión a los mecanismos de modelado desarrollados y que ahora hacen parte de la sintaxis de HLVL. Como se describió en la sección *Mecanismos de Modelado*, se propusieron distintos mecanismos sintácticos para soportar la escalabilidad de modelos y se definió una ruta de implementación. Esta ruta incluye el desarrollo de la Herencia e Interfaces, de las cuales, dadas las limitaciones de tiempo y alcance del proyecto, se desarrolló el mecanismo de Herencia y se adelantó un primer diseño de la sintaxis del mecanismo de Interfaces. Por lo tanto, se realizó la evaluación de ambos mecanismos, mediante una encuesta realizada a un conjunto de cuatro expertos.

2.1. PLANEACIÓN

En esta sección se presentan los elementos necesarios para llevar a cabo la evaluación del componente de modelado a través de una encuesta a expertos.

2.1.1. Criterios de selección de entrevistados

Debido a que la naturaleza del proyecto y sus implicaciones se encuentran sobre una rama de la ingeniería muy específica, los expertos seleccionados para responder la encuesta fueron escogidos teniendo en cuenta su alto nivel de entendimiento sobre la teoría de modelado de variabilidad y las SPL's. Teniendo en cuenta lo anterior y que la comunidad de expertos en SPL es reducida, la cantidad de expertos encuestados es pequeña. Sin embargo, cabe resaltar que se cumple con la triangulación, es decir, se logró la combinación de múltiples fuentes de datos para realizar el análisis.

2.1.2. Cuestionario

El cuestionario consta de: (i) una introducción que permite contextualizar al experto sobre el proyecto; (ii) explicaciones breves que ejemplifican los conceptos de herencia e interfaces para ofrecer mayor claridad al momento de responder las preguntas; (iii) preguntas que evalúan en qué medida las sintaxis propuestas representan los conceptos presentados y (iv) un consentimiento en el cual el experto pueda indicar la forma en que quiere que la información suministrada sea usada. Como anexo se dejan capturas de la Encuesta Soporte de Escalabilidad para HLVL con el detalle de cada uno de los puntos

descritos anteriormente. La encuesta puede ser encontrada en el siguiente enlace: <https://forms.gle/3nkDNmj8UZwpstzg6>

2.2. ANÁLISIS Y RESULTADOS

En esta sección se presenta el análisis y resultados de las respuestas obtenidas a través de la encuesta realizada a expertos.

2.2.1. Herencia

Pregunta 1: Esta pregunta expone tres propuestas de sintaxis para declarar la Herencia de modelos en HLVL, las palabras reservadas `extends`, `inherits` y el símbolo `:`. Como se puede observar en la Ilustración 32, tres de los cuatro expertos estuvieron de acuerdo con que `extends` es la opción que mejor representa el concepto de modelos heredados.



Ilustración 28. Pregunta 1 sección Herencia, encuesta soporte HLVL.

El experto que escogió la palabra reservada `inherits`, se justifica en que las demás opciones son usadas por otros lenguajes y podrían ocasionar confusiones. Por otro lado, los demás expertos, justifican que la palabra reservada `extends` es más sencilla de entender ya que por su traducción en inglés entienden que el nuevo modelo es una

extensión al que se agregan elementos de los modelos heredados. Una de las recomendaciones que se podría tomar en cuenta como trabajo a futuro es el uso de un alias al momento de declarar el modelo heredado en lugar del nombre del archivo. Esto implicaría implementar la sintaxis descrita en la Ilustración 17, la cual permite importar modelos.

Pregunta 2: Esta pregunta busca validar si la sintaxis: `<model-id> "." <element-id>`, permite o no representar la referencia a elementos de modelos heredados. Como puede verse en la Ilustración 33, tres de los cuatro expertos respondieron afirmativamente.

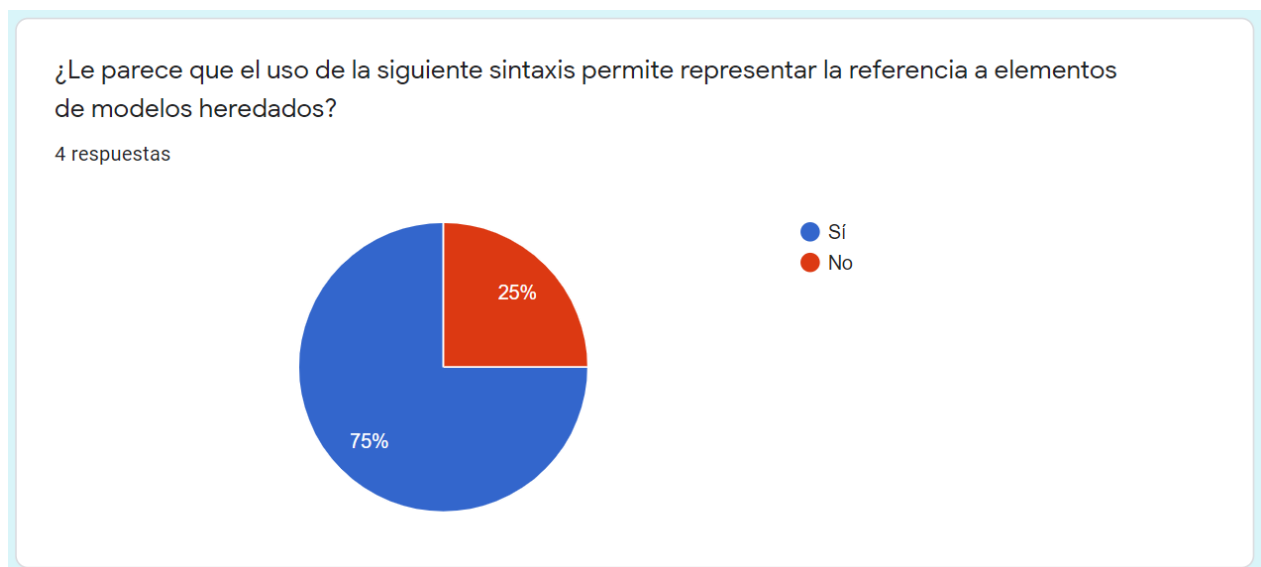


Ilustración 29. Pregunta 2 sección Herencia, encuesta soporte HLVL.

El experto que respondió negativamente justifica su respuesta en que el uso de la sintaxis no es completamente claro al identificar que los elementos son del modelo heredado. Por otro lado, los demás expertos justifican que les parece más cómodo el acceso a los elementos de un modelo mediante un punto debido a su similitud con otros lenguajes, por ejemplo, Java. Sin embargo, se hace notar que no hay una coherencia entre el identificador para declarar los modelos heredados (nombre del archivo) y el identificador usado para referenciar los elementos de dichos modelos.

2.2.2. Interfaces

Pregunta 1: Esta pregunta busca validar si la sintaxis propuesta en la Ilustración 17 permite o no representar el concepto de interfaces para un modelo. Como puede observarse en la Ilustración 34; **Error! No se encuentra el origen de la referencia.**, dos de los tres expertos respondieron afirmativamente.

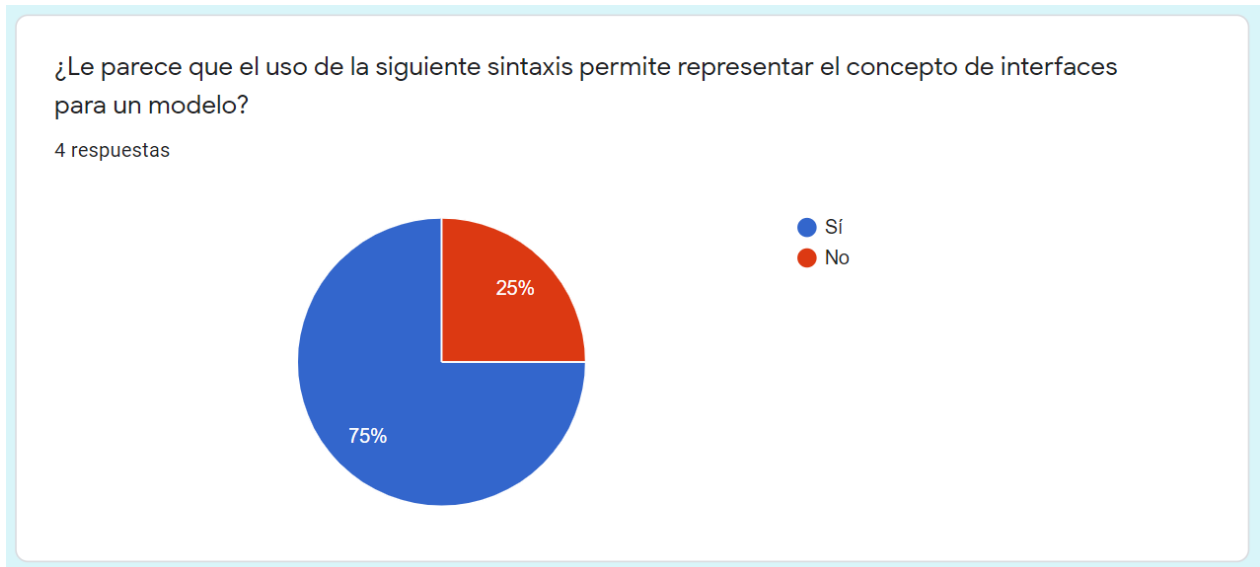


Ilustración 30. Pregunta 1 sección interfaces, encuesta soporte HLVL.

El experto que respondió negativamente justifica que no ve necesario el uso de interfaces si ya se dispone del mecanismo de herencia. Propone en lugar de utilizar interfaces hacer uso de clases abstractas, esto podría significar que las interfaces no sean declaradas dentro del mismo modelo sino en un archivo independiente a manera de clase abstracta para que fuera heredada por el modelo. Los demás expertos justifican su respuesta afirmativa en que es clara y similar a otros lenguajes.

Pregunta 2: Esta pregunta busca validar si la sintaxis propuesta en la Ilustración 18 permite o no representar el uso de una interfaz dentro de otro modelo. Como puede observarse en la Ilustración 35; **Error! No se encuentra el origen de la referencia.**, las respuestas estuvieron divididas por igual.

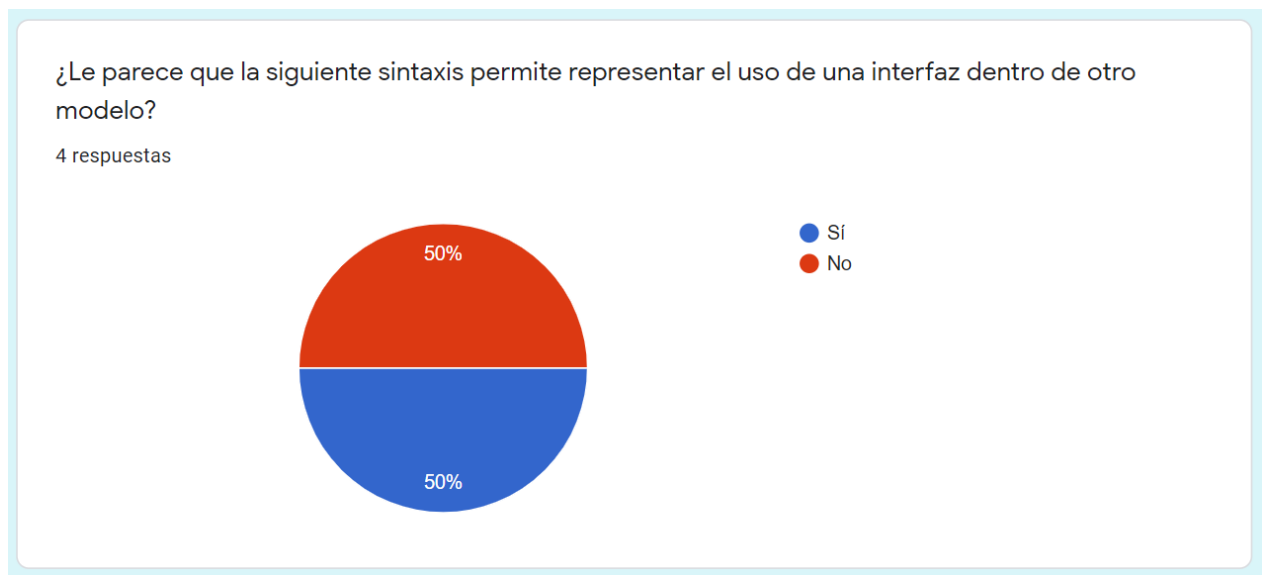


Ilustración 31. Pregunta 2 sección interfaces, encuesta soporte HLVL.

Los expertos que respondieron afirmativamente justifican su respuesta con que la propuesta es similar a otros lenguajes y permite la selección del modelo y la interfaz deseada de forma coherente. Los demás expertos justifican su respuesta negativa en que no proporciona claridad suficiente sobre el significado semántico de la sintaxis, además uno de los expertos afirma que el mecanismo parece más un sistema de permisos que el uso de una interfaz, por lo que sería suficiente el definir atributos con diferentes alcances (public/protected/private).

2.3. CONCLUSIONES

Tomando en cuenta los resultados de la evaluación podemos concluir lo siguiente para el componente de modelado de la extensión desarrollada: en cuanto al mecanismo de Herencia podemos confirmar que las propuestas de sintaxis desarrolladas representan de la mejor forma el concepto de Herencia dado que son sencillas de entender e intuitivas por su semejanza con otros lenguajes como Java. Sin embargo, hace falta coherencia entre el identificador para declarar el modelo Heredado y el usado para referenciar sus elementos. Además, como trabajo a futuro se podría encontrar valor en integrar el mecanismo de Herencia con Interfaces para que los modelos sean primero importados mediante el nombre del archivo del modelo y posteriormente heredados a través de un alias, el cual podría ser el nombre del modelo, su elemento raíz o un identificador que permita darle mayor entendimiento al usuario. En el caso del mecanismo de interfaces podemos afirmar que es necesario mejorar su entendimiento a nivel semántico para que sea más clara su relevancia como mecanismo de modularización. La propuesta de sintaxis para declarar interfaces dentro de un modelo permite representar el concepto de Interfaces, pues es similar a otros lenguajes, sin embargo, es posible que el mecanismo sea objeto de cambios en el futuro pues existen confusiones sobre algunos aspectos como su pertinencia, utilidad además de algunas propuestas de mejora.

3. Componente de análisis

El componente de análisis de la extensión hace alusión a las operaciones de análisis desarrolladas y que ahora hacen parte de los servicios que se ofrecen para HLVL. Como se describió en la sección *Operaciones de Análisis*, se propusieron distintas operaciones para soportar la escalabilidad de modelos y se definió una ruta de implementación. Esta ruta incluye el desarrollo de las operaciones de Agregación y *Merge*, las cuales fueron evaluadas a través de una serie de casos de estudio. El objetivo de dichas evaluaciones es validar las características de correctitud y escalabilidad del componente de análisis. En el caso de la correctitud, se compararon los resultados entre las operaciones para HLVL y el lenguaje FAMILIAR de modo que, si se obtiene la misma cantidad de soluciones en ambos modelos resultantes, se puede afirmar que las operaciones son correctas. En el caso de la escalabilidad, se busca validar que la extensión permita el modelado de líneas de productos usados en la industria. Para esto se seleccionó un caso de estudio basado en la industria automotriz propuesto por Chavarriaga, (2017) el cual permite observar la especificación de diferentes subsistemas y la interacción entre estos. Este caso de estudio se presenta en la sección *Casos de estudio*; **Error! No se encuentra el origen de la referencia..**

3.1. PLANEACIÓN

En esta sección se presentan los elementos necesarios para llevar a cabo la evaluación del componente de análisis a través de casos de estudio.

3.1.1. Criterios de selección

Para la selección de los casos de estudio se tuvo en cuenta: (i) que sean modelos de características, ya que la extensión está restringida al uso de este tipo de modelos; (ii) que los modelos tuvieran una cantidad mínima de características, ya que se busca evaluar modelos de diferentes tamaños; (iii) que la línea de productos modelada sea fácil de reconocer, con el objetivo de facilitar el entendimiento del modelo y (iv) en lo posible, que modelen productos reales.

3.1.2. Recolección de datos

Para la recolección de los casos de estudio utilizados para evaluar la característica de correctitud de las operaciones de Agregación y *Merge*, se seleccionaron algunos de los

modelos encontrados en el repositorio oficial de FeatureIDE. Por otro lado, para evaluar la característica de escalabilidad de la extensión, se seleccionó un modelo basado en una línea de automóviles. Este modelo es ideal ya que pertenece a un dominio que ha sido ampliamente estudiado en la literatura y puede ser más amigable para el entendimiento de personas no expertas (Chavarriaga, 2017).

3.1.3. Métrica de evaluación

La métrica usada en la evaluación del componente de análisis es el número de soluciones o configuraciones de los diferentes modelos usados en los casos de estudio. Dado que buscan comparar las métricas para HLVL y FAMILIAR, se deben seguir dos procesos distintos para obtener el número de soluciones de los modelos. En el caso de HLVL, el proceso a seguir es: (i) realizar la transformación del modelo en el lenguaje FeatureIDE a HLVL, (ii) obtener el CNF en formato DIMACS del modelo que sirve como input para el siguiente paso, (iii) utilizar un SAT solver para obtener el número de soluciones del modelo, el solver utilizado fue SAT4J. En el caso de FAMILIAR, el proceso a seguir es: (i) realizar la transformación del modelo en el lenguaje FeatureIDE a FAMILIAR, (ii) construir un script en FAMILIAR en el que se declare el modelo, (iii) declarar en el script el uso de la operación *countings*, con la cual se obtiene el número de soluciones del modelo.

3.1.4. Casos de estudio

Dado que las operaciones de Agregación y *Merge* se aplican sobre modelos independientes y superpuestos respectivamente, es necesario modificar los modelos de los casos de estudio para que cumplan con las características dependiendo de la operación a evaluar. En el caso de la Agregación, se dividió el modelo en modelos independientes correspondientes a cada uno de los hijos de la raíz y en el caso del *Merge* se duplicó el modelo y se eliminaron características distintas en cada uno, de forma que en los modelos resultantes quedaran características equivalentes, es decir, con el mismo nombre.

Caso de estudio: *Toy Model*

El modelo *Toy Model*, es un modelo pequeño de prueba que no representa una línea de productos real, creado para evaluar el comportamiento de las operaciones de Agregación y *Merge* en HLVL.

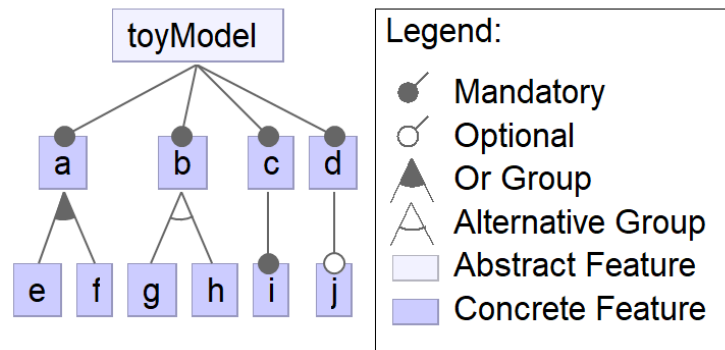


Ilustración 32. Caso de estudio *Toy Model*.

Caso de estudio: FameDB

El modelo FameDB, es un modelo mediano de prueba que representa la línea de base de datos de series FAME, desarrollada por SunGard (Howson, 2019). Este modelo fue escogido para evaluar el comportamiento de las operaciones de Agregación y *Merge* en HLVL.

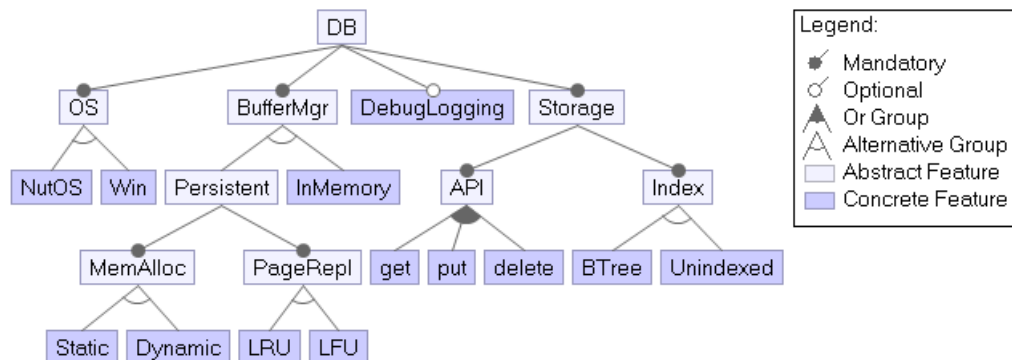


Ilustración 33. Caso de estudio FameDB

Caso de estudio: GPL

El modelo GPL, es un modelo grande de prueba que representa una familia de aplicaciones de grafos clásicas propuesta por Lopez-Herrejon & Batory, (2001) cuyo objetivo es evaluar

metodologías sobre líneas de productos. Este modelo fue escogido para evaluar el comportamiento de la operación de Agregación en HLVL.

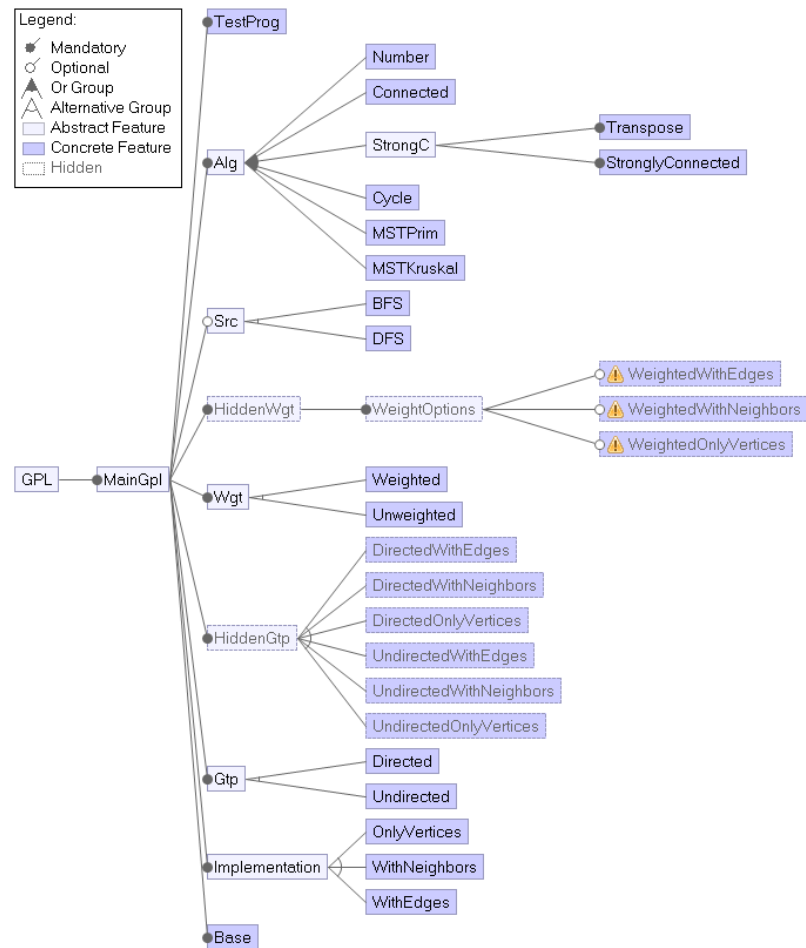


Ilustración 34. Caso de estudio GPL

Caso de estudio: *CarSystem*

El modelo *CarSystem*, es un modelo grande de prueba que representa una familia de automóviles, propuesto por (Chavarriaga, 2017). Este modelo fue escogido para evaluar el comportamiento de la operación de *Merge* en HLVL.

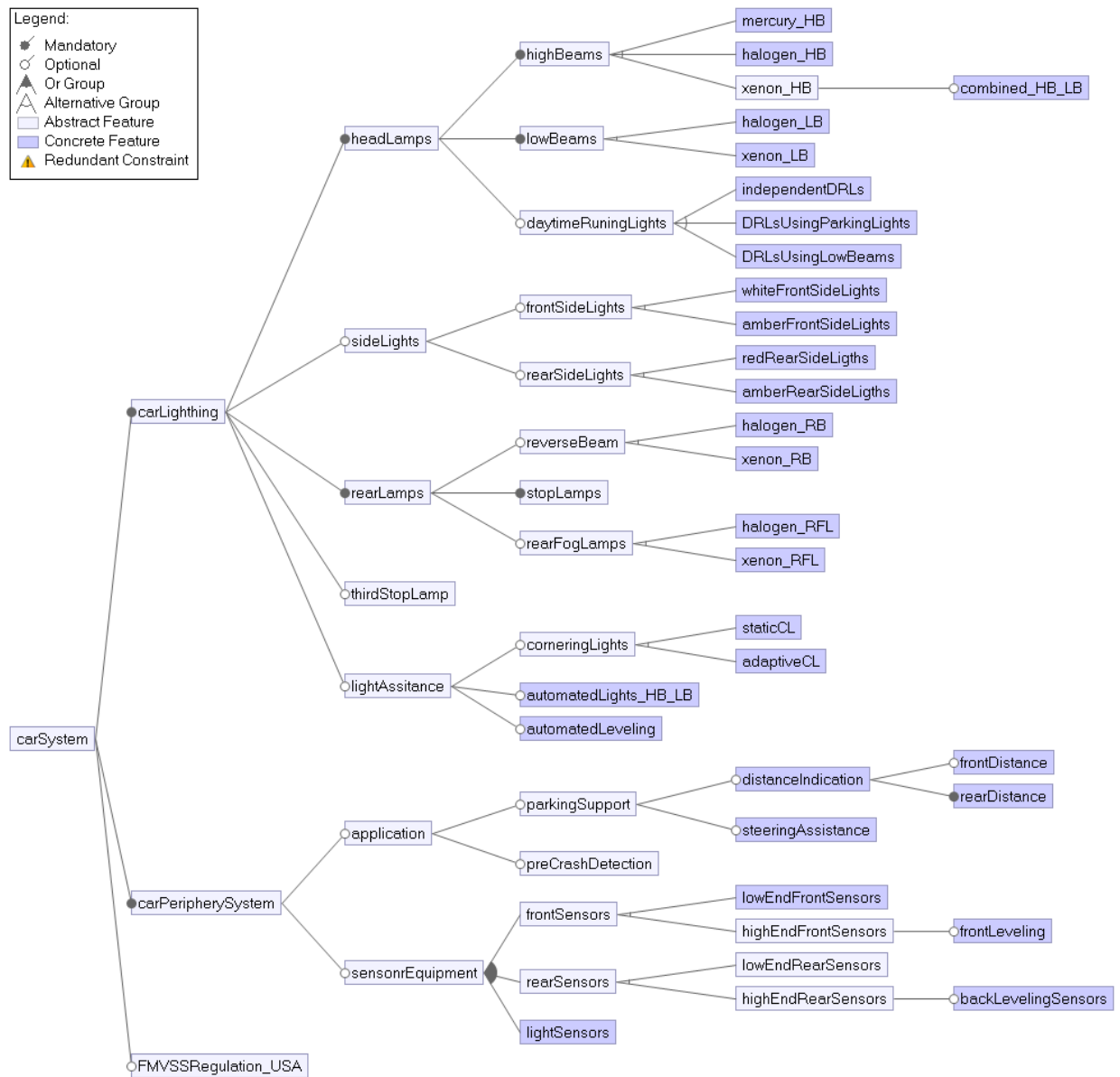


Ilustración 35. Caso de estudio *CarSystem*

3.2. ANÁLISIS Y RESULTADOS

A continuación, se presentan los resultados de la ejecución de las pruebas de evaluación para las operaciones de Agregación (Tabla 14) y *Merge* (Tabla 15; **Error! No se encuentra el origen de la referencia.**).

Tabla 14. Resultados evaluación Agregación

	Número de Soluciones	
Caso de estudio	FAMILIAR	HLVL
Toy Model	12	12
FameDB	140	140
GPL	72576	72576

Tabla 15. Resultados evaluación Merge

	Número de Soluciones	
Caso de estudio	FAMILIAR	HLVL
Toy Model	5	5
FameDB	54	54
CarSystem	2400	2400

3.3. CONCLUSIONES

Como puede observarse, para ambas operaciones el número de soluciones obtenido es el mismo. Según la métrica de evaluación definida, esto indica que las operaciones de Agregación y de *Merge* fueron implementadas correctamente.

Algunas de las limitaciones encontradas al momento de evaluar el componente de análisis fueron: (i) el *solver* utilizado para obtener el número de soluciones no permitía obtener las configuraciones concretas para los modelos, lo cual no permitió hacer otro tipo de validaciones sobre las operaciones, (ii) para modelos muy grandes el *solver* no era capaz de obtener el número de exacto de soluciones, solamente un valor aproximado, (iii) FAMILIAR no fue capaz de procesar modelos muy grandes, lo que limitó la selección a modelos de menor tamaño.

CAPÍTULO 6: Conclusiones y Trabajo a Futuro

1. Resultados Obtenidos

Este capítulo se centra en presentar los resultados, contribuciones, entregables y conclusiones producto del desarrollo del proyecto. En cuanto a los aportes, se pueden encontrar dos tipos, los que están relacionados con los objetivos del proyecto, es decir, los artefactos generados para dar solución del problema y los que están relacionados con el desarrollo de capacidades del investigador, es decir, los que aportan a su crecimiento profesional. En cuanto a los resultados, se presenta el listado de los entregables obtenidos en el desarrollo de cada uno de los objetivos específicos.

1.1. APORTES RELACIONADOS CON EL OBJETO DEL PROYECTO

El desarrollo de este proyecto contribuyó con los siguientes aportes:

1. Mejorar el soporte para la escalabilidad de modelos especificados en HLVL, mediante una extensión que le permita especificar y hacer análisis de modelos de variabilidad fraccionados. Además de demostrar por medio de una evaluación de expertos que tan adecuada es la propuesta de la extensión.
2. Revisión de literatura, definición y clasificación de técnicas y operaciones para el soporte de escalabilidad en lenguajes para el modelado de variabilidad.
3. Síntesis comparativa sobre las técnicas y operaciones para el soporte de escalabilidad en lenguajes para el modelado de variabilidad.
4. Entregar un documento del proyecto de grado que podrá ser de contribución y utilidad para la comunidad académica.

1.2. APORTES RELACIONADOS CON EL DESARROLLO DE CAPACIDADES DEL INVESTIGADOR

La planeación y desarrollo de este proyecto tendrá los siguientes aportes a las capacidades de los investigadores:

1. Desarrollo de habilidades de investigación:
 - a. Búsqueda de información en fuentes académicas y de dominio específico
 - b. Comunicación oral efectiva de ideas.
 - c. Escritura académica.
 - d. Pensamiento crítico y reflexivo.
2. Conocimientos adquiridos sobre teoría y práctica de conceptos relacionados a la Ingeniería de Líneas de productos de *Software*.
3. Experiencia adquirida en el desarrollo de proyectos que no implican únicamente el desarrollo de *software* sino también un componente investigativo.

1.3. RESULTADOS Y ENTREGABLES

Los entregables se listan a continuación señalando el objetivo específico al que están asociados:

OE1: Seleccionar un conjunto de técnicas y/u operaciones que permitan soportar la escalabilidad de modelos especificados en HLVL.

- Reporte de escenarios que definen las técnicas y operaciones que soportan escalabilidad en los lenguajes de modelado de variabilidad.
- Diseño de prototipos para las operaciones y técnicas para el soporte de escalabilidad.
- Sección del documento correspondiente.
- Diapositivas de la presentación del proyecto.

OE2: Desarrollar una extensión del lenguaje HLVL que permita soportar la escalabilidad de modelos.

- Capítulo del diseño de la extensión del lenguaje.
- Repositorios con la implementación de los componentes de modelado y análisis de la extensión.
- Diapositivas de la presentación del proyecto.

- Repositorio con el código de la extensión.

OE3: Evaluar los componentes de modelado y análisis de la extensión.

- Capítulo con la evaluación de la extensión:
 - Componentes de modelado.
 - Formulario de la encuesta realizada a expertos.
 - Componente de análisis.
 - Diseño de los casos de estudio
 - Análisis y resultados de la evaluación
- Repositorio con la implementación de las pruebas para el componente de análisis
 - Modelos en HLVL
 - *Scripts* en FAMILIAR
- Diapositivas de la presentación del proyecto.

2. Conclusiones

El trabajo realizado durante el proyecto consistió en el desarrollo de una extensión del lenguaje HLVL para modelado de variabilidad, la cual permite soportar la escalabilidad a través de la composición de modelos. Esto fue alcanzado implementando el mecanismo sintáctico de Herencia y las operaciones Agregación y *Merge*.

El objetivo general del proyecto de desarrollar mejoras para el soporte de escalabilidad de HLVL, implicó un trabajo extenso tanto de investigación como de implementación. Sin embargo, el objetivo se cumplió y dentro de las mejoras desarrolladas, correspondientes a los objetivos específicos se encuentran:

Un reporte detallado de las técnicas y operaciones que permiten soportar la escalabilidad en lenguajes de variabilidad, a través de las características de composición y modularización. En este reporte se especifica detalladamente un abanico de propuestas con las que un lenguaje podría soportar la escalabilidad. Esto se realizó como un primer paso para tener claridad de las posibles soluciones que se podrían desarrollar para dar soporte de escalabilidad a HLVL. Con lo anterior se cumplió con el primer objetivo específico.

La extensión del lenguaje, la cual consta de un componente de modelado, en el que se implementó el mecanismo de Herencia y un componente de análisis en el que se implementaron las operaciones de Agregación y *Merge*. Lo anterior implicó realizar cambios directos sobre la gramática del lenguaje e implementar microservicios que permitieran manipular los modelos. Con lo anterior se cumplió con el segundo objetivo específico.

La respectiva evaluación de los componentes de modelado y análisis de la extensión. En el caso del componente de modelado, la evaluación del mecanismo de Herencia tuvo como resultado que las propuestas de sintaxis desarrolladas, en efecto, representan de la mejor forma el concepto de Herencia gracias a su sencillez y similitud con otros lenguajes de programación. La evaluación del mecanismo de interfaces hizo evidente una deficiencia en el desarrollo semántico de la propuesta. Dado que no fue posible su implementación, sólo se desarrolló como una propuesta de sintaxis y es necesario profundizar en sus implicaciones semánticas para evaluar su idoneidad sintáctica.

Dentro de los mayores aprendizajes que se obtuvieron se encuentra:

- El desarrollo de habilidades de investigación como aprender a realizar revisiones literarias, comunicar efectivamente las ideas de forma oral y escrita.
- La adquisición de conocimientos teóricos y prácticos en el área de la SPLE.
- El desarrollo de habilidades necesarias para la construcción de lenguajes de dominio específico.
- La experiencia en el desarrollo de proyectos con un gran componente investigativo.

En cuanto a las limitaciones, se puede resaltar que la mayoría de las dificultades se presentaron debido a que la implementación del proyecto ya había sido empezada, a que se tenía desconocimiento del área y del *stack* tecnológico usado para el trabajo de construcción de lenguajes y modelamiento y análisis de SPLs. Además, el alcance de este proyecto se vio limitado por su gran extensión, restricciones de tiempo y la situación excepcional producto de la pandemia. Es por esto, que existen grandes posibilidades para el trabajo a futuro, de las cuales se proponen dos componentes principales.

Extensión

En el CAPÍTULO 3: Escenarios para el Soporte de Escalabilidad de HLVL se proponen seis mecanismos de modelado, de los cuales se implementó únicamente el mecanismo de Herencia, se realizó la evaluación del mecanismo de Interfaces y se implementaron las operaciones de análisis de Agregación y *Merge*. Como resultado de la evaluación del componente de modelado, se obtuvo una posibilidad de mejora para el mecanismo de Herencia. Esta mejora consiste en integrar la Herencia con el mecanismo de Interfaces de forma que se pueda heredar un modelo previamente importado, mediante un alias. Lo anterior solucionaría el problema que se genera al momento de usar identificadores distintos para declarar un modelo heredado y declarar el acceso a sus elementos.

Integración con *framework*

Actualmente, existe un *framework* que implementa HLVL llamado [*Coffee*](#), el cual ofrece diferentes servicios para la especificación y análisis de modelos de variabilidad. A futuro, se plantea la posibilidad de integrar la extensión de HLVL con *Coffee*. Para esto, se deja una primera versión de la arquitectura del sistema en los Anexos. Debido a que *Coffee*

presenta una arquitectura basada en microservicios, se plantea adicionar la extensión como otro microservicio.

Anexos

Lista de Acrónimos

BDD	Diagramas Binarios de Decisión (<i>Binary Decision Diagram</i>)
CSP	Problema de Satisfacción de Restricciones (<i>Constraint Satisfaction Problem</i>)
FAMILIAR	Lenguaje de scripting para la manipulación y razonamiento sobre modelos de características (<i>FeAture Model script Language for manipulation and Automatic Reasoning</i>)
FODA	Análisis de dominio orientado a características (<i>Feature Oriented Domain Analysis</i>)
GPS	Sistema de posicionamiento global (<i>Global Positioning System</i>)
HLVL	Lenguaje de variabilidad de alto nivel (<i>High Level Variability Language</i>)
IVML	Lenguaje Integrado para el Modelado de Variabilidad (<i>Integrated Variability Modeling Language</i>)
MP3	Capa de audio MPEG-1 3 (<i>MPEG-1 Audio Layer 3</i>)
OTAN	Organización del Tratado del Atlántico Norte
SAT	Problema de Satisfacibilidad Booleana (<i>Boolean Satisfiability Problem</i>)
SPL	Líneas de productos de <i>software</i> (<i>Software Product Lines</i>)
SPLC	Conferencia de líneas de productos de <i>software</i> (<i>Software Product Lines Conference</i>)
SPLE	Ingeniería de líneas de productos de <i>software</i> (<i>Software Product Lines Engineering</i>)
TVL	Lenguaje de Variabilidad Textual (<i>Textual Variability Problem</i>)
VSL	Lenguaje de Especificación de Variabilidad (<i>Variability Specification Language</i>)
CNF	Forma Normal Conjuntiva (<i>Conjunctive Normal Form</i>)
EBNF	Forma Extendida de Backus-Naur (<i>Extended Backus-Naur Form</i>)
DIMACS	Centro para las Matemáticas Discretas y Ciencias de la Computación Teóricas (Center for D iscrete M athematics and Theoretical C omputer S cience)

Glosario de Términos

GPS.

Sistema de posicionamiento global que permite determinar en toda la tierra la posición de cualquier objeto con una precisión de hasta centímetros (Fundación Wikipedia Inc, 2020).

High-Level Variability Language (HLVL).

Lenguaje que permite describir la variabilidad de líneas de productos en diferentes enfoques como el orientado a características, a puntos de variación y a decisión (Villota et al., 2019).

Lenguaje FODA.

Lenguaje para el modelado de variabilidad que soporta la reutilización de productos, mediante la representación de sus funcionalidades por medio de características (K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, 1990).

Modelo de Objetivos (*Goal Model*).

Representación de requerimientos e intenciones de un sistema de *software*. En el modelo se describen intenciones, metas, tareas y relaciones con contribuciones negativas y positivas. Estos modelos guían el desarrollo de la variabilidad en las SPL's ya que proveen la fuente de la variabilidad (Anderson & Fickas, 1989).

Modelo de Variabilidad.

Especificación de los elementos comunes y variables de un sistema, las diferentes variantes de esos elementos y las restricciones sobre dichas variantes.

MP3.

Formato de compresión de audio digital que usa un algoritmo con pérdida para conseguir un menor tamaño de archivo (Fundación Wikipedia Inc, 2020).

Ingeniería de Líneas de Productos de *Software* (SPLE).

Paradigma para el desarrollo de aplicaciones de *software* (sistemas intensivos de *software* y productos de *software*) mediante el uso de plataformas y personalización en masa (Pohl et al., 2005)

Stakeholder.

Individuo, grupo u organización que puede afectar, ser afectada por las decisiones, actividades o resultados de un proyecto (Fundation Wikipedia Inc, 2020).

Variabilidad.

Capacidad o tendencia de algo a cambiar, dentro de la ingeniería de productos de *software* se refiere a los cambios que son introducidos dentro de un sistema, para producir sistemas diferentes (Pohl et al., 2005).

Solver.

Herramienta de *software* que toma las descripciones de un problema de satisfacibilidad booleana o de restricciones y calcula su solución (Fundación Wikipedia Inc, 2019).

Ad-hoc.

Referente a una solución que ha sido elaborada específicamente para un problema o fin preciso, por tanto, no es generalizable ni utilizable con otros propósitos (Fundación Wikipedia Inc, 2020).

Análisis de participación

Beneficiarios directos

- Ángela Villota

Beneficiarios indirectos

- Empresas que manejan productos por características, que busquen producirlos en masa y administrar sus familias de productos. Por ejemplo, la industria automotriz.
- Comunidad de ingeniería de líneas de productos.
- Miembros de la comunidad, quienes estén desarrollando un estándar.

Árbol del Problema

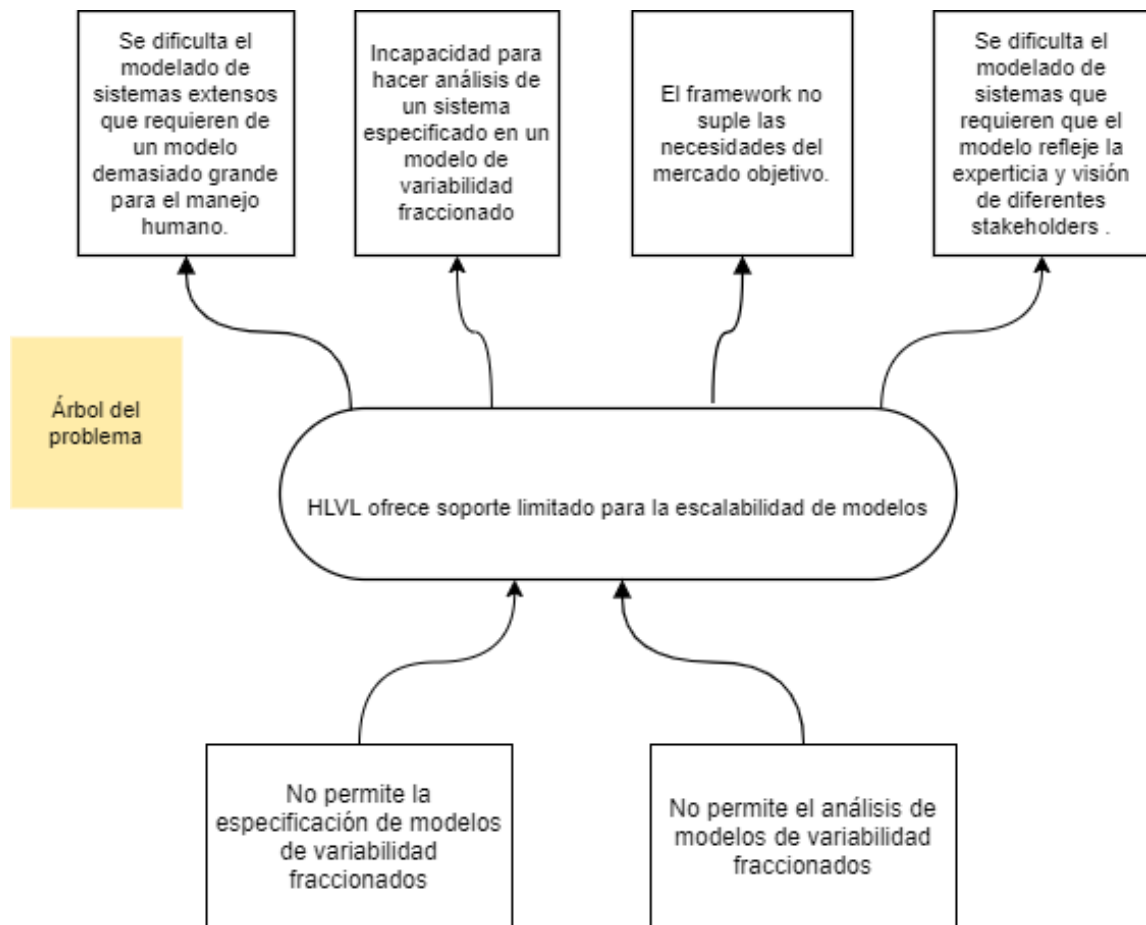


Ilustración 36. Árbol del problema

Árbol de Objetivos

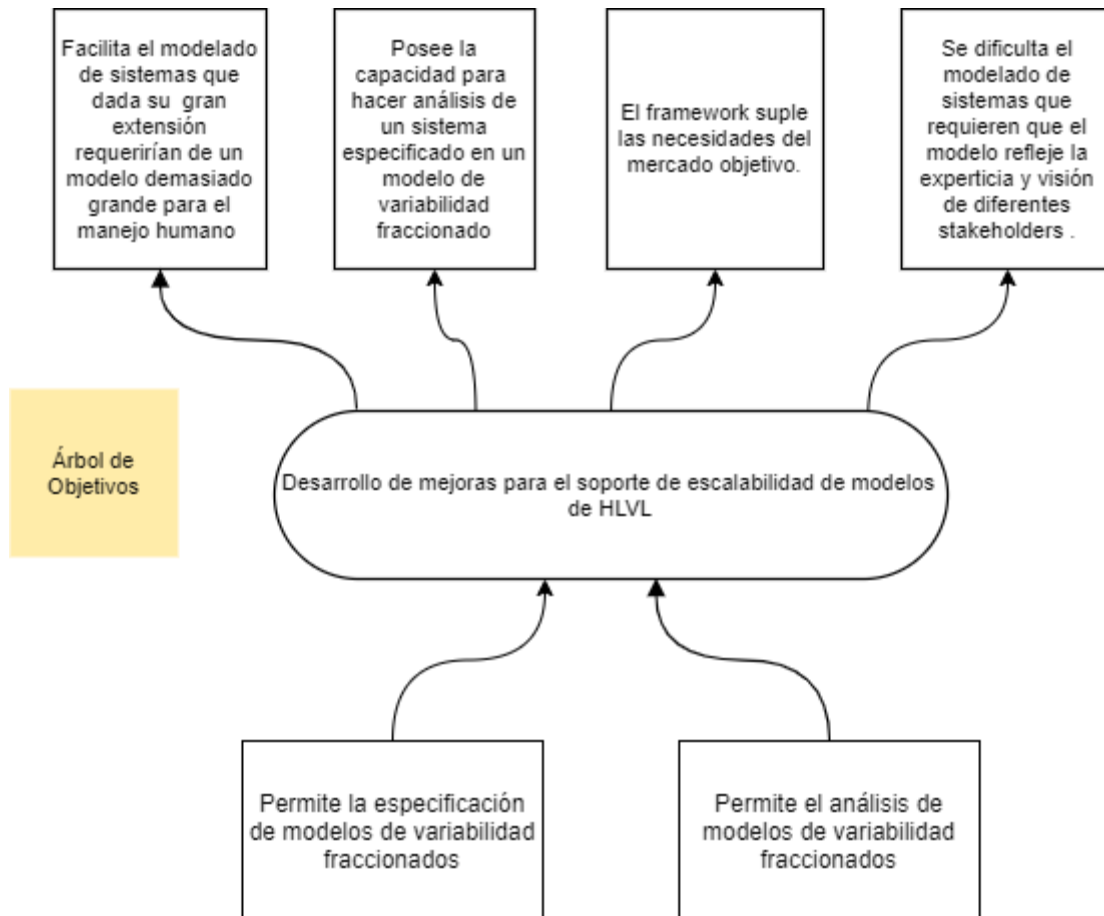
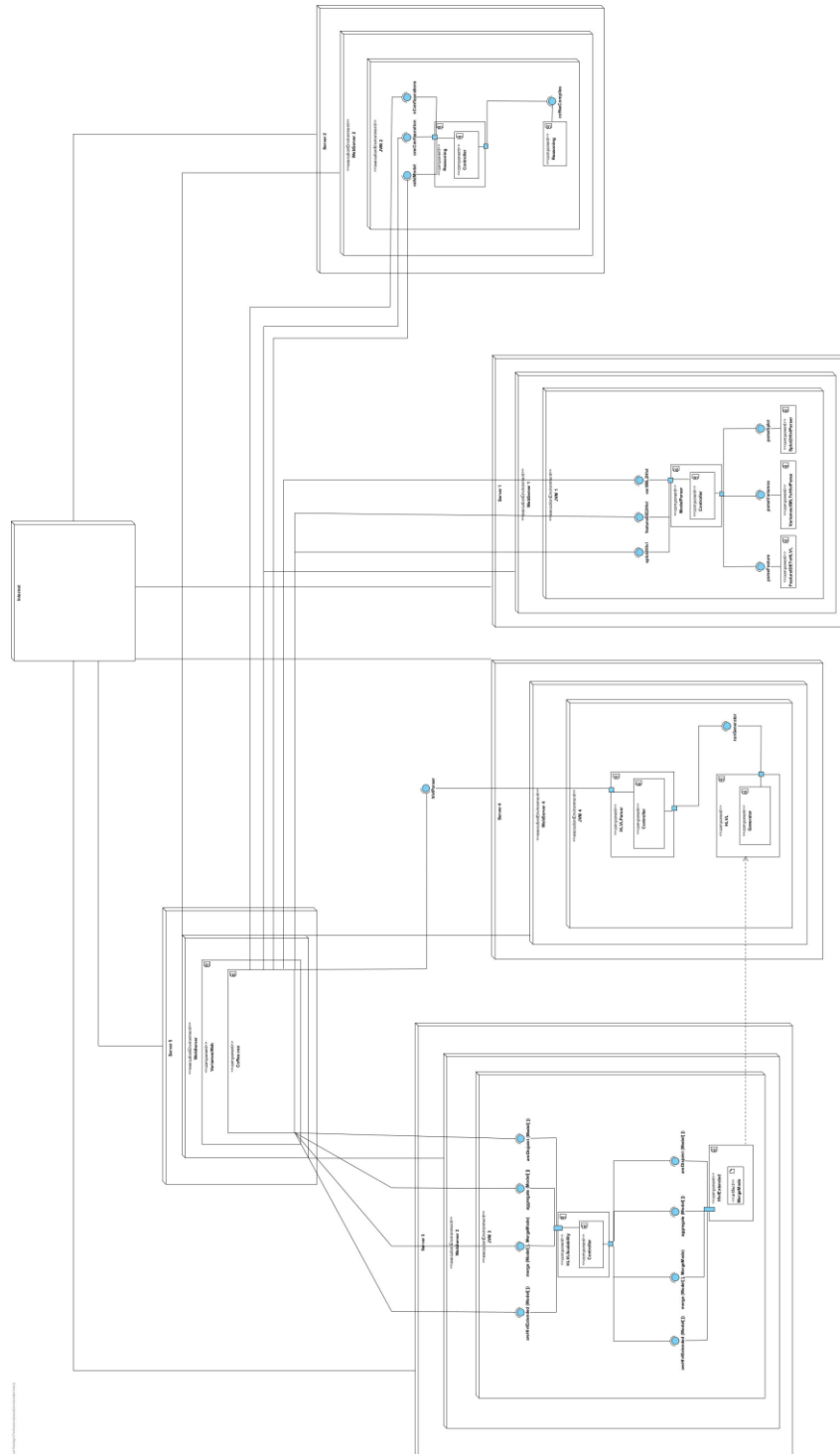


Ilustración 37. Árbol de objetivos

Propuesta de Arquitectura - Integración de la Extensión de HLVL con COFFEE



Encuesta Soporte de Escalabilidad para HLVL

PREGUNTAS MECANISMO HERENCIA

Soporte de escalabilidad para HLVL

El proyecto de grado "Extendiendo HLVL para la especificación y el análisis de modelos de variabilidad fraccionados" busca que el lenguaje de variabilidad HLVL cuente con mecanismos de escalabilidad en cuanto a composición y modularización de modelos. El objetivo de esta encuesta es evaluar las propuestas de sintaxis de los mecanismos de Herencia e Interfaces, las cuales permitirían a HLVL ofrecer soporte para composición y modularización respectivamente.

*Obligatorio

Mecanismo de Herencia

La característica de composición hace referencia al proceso en el cual se combinan diferentes modelos de variabilidad para formar un único modelo. Esto permite especificar diferentes subsistemas y dominios de aplicación de la línea de productos en diferentes modelos de variabilidad. En este contexto, se propone el mecanismo de herencia como una forma de soportar la característica de composición. Este mecanismo permite que un modelo haga referencia a elementos de modelos externos especificando un conjunto de modelos de los cuales hereda.

Por ejemplo, supongamos que tenemos el modelo A con los elementos o, p y q, y el modelo B con los elementos r y s, donde B hereda de A. En este caso A es el modelo heredado, es decir, le permite a B acceder a sus elementos para referenciarlos al momento de definir restricciones. Ahora el modelo B tiene acceso a los elementos o, p y q del modelo A.

¿Cuál de las siguientes palabras reservadas o símbolos considera que mejor representa el concepto de modelos heredados? *

```
model PC extends "Procesador.hlvl", "SistemaOperativo.hlvl"
elements:
  boolean pc
  boolean tarjetaGrafica
  boolean msi
  boolean asus
  boolean evga
relations:
  com1: common(pc)
  grp2: group(tarjetaGrafica, [msi, asus, evga],[1,1])
```

```
model PC inherits "Procesador.hlvl", "SistemaOperativo.hlvl"
elements:
  boolean pc
  boolean tarjetaGrafica
  boolean msi
  boolean asus
  boolean evga
relations:
  com1: common(pc)
  grp2: group(tarjetaGrafica, [msi, asus, evga],[1,1])
```

```
model PC : "Procesador.hlvl", "SistemaOperativo.hlvl"
elements:
  boolean pc
  boolean tarjetaGrafica
  boolean msi
  boolean asus
  boolean evga
relations:
  com1: common(pc)
  grp2: group(tarjetaGrafica, [msi, asus, evga],[1,1])
```

☐ Palabra reservada 'extends'

☐ Palabra reservada 'inherits'

☐ Símbolo ":"

☐ Otros: _____

Por favor, en este espacio justifique por qué escogió la respuesta anterior. *

Tu respuesta _____

¿Tiene alguna recomendación respecto a las propuestas de sintaxis para especificar modelos heredados?

Tu respuesta _____

Referenciación de elementos de modelos heredados

Para poder hacer uso de un elemento de un modelo heredado, es necesario una sintaxis que permita especificar un identificador para dicho elemento.

¿Le parece que el uso de la siguiente sintaxis permite representar la referencia a elementos de modelos heredados? *

```
model PC : "Procesador.hlvl", "SistemaOperativo.hlvl"
elements:
  boolean pc
  boolean tarjetaGrafica
  boolean msi
  boolean asus
  boolean evga
relations:
  com1: common(pc)
  grp1: group(tarjetaGrafica, [msi, asus, evga], [1,1])
  mut1: mutex(Procesador.dosNucleos, evga)
  imp1: implies(SistemaOperativo.windows, msi)
```

☐ Sí

☐ No

Por favor, indique por qué escogió la respuesta anterior. *

Tu respuesta _____

¿Tiene alguna recomendación respecto a la propuesta de sintaxis para especificar la referenciación a elementos de modelos heredados?

Tu respuesta _____

PREGUNTAS MECANISMO INTERFACES

Mecanismo de Interfaces

La característica de modularización hace referencia a la medida en que el lenguaje de variabilidad permite descomponer los elementos dentro un modelo para ser tratados como entidades independientes. En este contexto, se propone el mecanismo de interfaces como una forma de soportar la característica modularización. Este mecanismo se utiliza para limitar la visibilidad de los elementos de un modelo al momento de ser heredado por otro modelo. Cuando un modelo hereda de otro modelo, obtiene acceso a todos sus elementos, sin embargo, cuando hereda de la interfaz de otro modelo, obtiene acceso a un subconjunto previamente definido de sus elementos.

Por ejemplo, supongamos que tenemos el modelo A, con los elementos o, p y q y la interfaz K que contiene los elementos o y p, y el modelo B que tiene los elementos r y s, donde B hereda de la interfaz K. En este caso, el modelo B tiene acceso a los elementos o y p del modelo A que son los definidos en la interfaz K.

¿Le parece que el uso de la siguiente sintaxis permite representar el concepto de interfaces para un modelo? *

```
model Procesador
  interfaces:
    gamaAlta exports [cuatroGHz, N4, H3]
    gamaMedia exports [dosGHz, N3, H2]
    gamaBaja exports [unoGHz, N2, H2]
  elements:
    boolean procesador
    boolean frecuencia
    boolean unoGHz
    boolean dosGHz
    boolean cuatroGHz
    boolean nucleos
    boolean N2
    boolean N3
    boolean N4
    boolean hilos
    boolean H2
    boolean H3
  relations:
    com1: common(procesador)
    grp1: group(procesador, [frecuencia, nucleos, hilos], [3,3])
    grp2: group(frecuencia, [unoGHz, dosGHz, cuatroGHz], [1,1])
    grp3: group(hilos, [H2, H3], [1,1])
    grp4: group(nucleos, [N2, N3, N4], [1,*])
    mut1: mutex(unoGHz, H3)
    imp1: implies(N4, cuatroGHz)
```

- ☐ Sí
- ☐ No

Por favor, indique por qué escogió la respuesta anterior. *

Tu respuesta

¿Tiene alguna recomendación respecto a la propuesta de sintaxis para especificar la declaración de interfaces dentro de un modelo?

Tu respuesta

Uso de una interfaz

Para que se haga uso de una interfaz declarada en un modelo, es necesario que dicha interfaz sea referenciada en otro modelo a través de una herencia.

¿Le parece que la siguiente sintaxis permite representar el uso de una interfaz dentro de otro modelo? *

```
import "Procesador.hlv1"
model PC extends Procesador.gamaAlta
  elements:
    boolean pc
    boolean tarjetaGrafica
    boolean msi
    boolean asus
    boolean evga
  relations:
    com1: common(pc)
    grp1: group(tarjetaGrafica, [msi, asus, evga])[1,1]
    decl1: decomposition(pc, [tarjetaGrafica])[0,1]
    imp1: implies(msi, os.windows)
    imp2: implies(Procesador.cuatroGHz, evga)
    mut1: mutex(Procesador.N4, evga)
    mut2: mutex(Procesador.N3, asus) // Error: El elemento N3 no es accesible.
```

- ☐ Sí
- ☐ No

Por favor, indique por qué escogió la respuesta anterior. *

Tu respuesta

¿Tiene alguna recomendación respecto a la propuesta de sintaxis para especificar el uso de una interfaz dentro de otro modelo?

Tu respuesta

¿Tiene algún comentario respecto al grado en que las propuestas permiten soportar las características de modularización y composición de modelos?

Tu respuesta

PREGUNTAS CONSENTIMIENTO

Consentimiento para hacer uso de la información diligenciada en este formulario

Agradecemos su participación como experto porque su concepto enriquece nuestro trabajo. Nos gustaría incluir la información diligenciada en este formulario en el reporte del proyecto. Por favor, especifique cómo le gustaría que tratáramos la información que nos ha proporcionado.

¿Esta de acuerdo con que la información que haya proporcionado en este formulario sea usada en el reporte del proyecto? *

- ☐ Permiso que mis comentarios sean citados con mi nombre
- ☐ Permiso que se haga uso de mis comentarios anónimamente

Si su respuesta a la pregunta anterior lo requiere, por favor escriba su nombre.

Tu respuesta

Análisis de riesgos y limitaciones

Se identificaron ocho riesgos que podrían impactar el desarrollo del proyecto. A continuación, se presentan los riesgos, su impacto, probabilidad y planes de contingencia.

Tabla 16. Matriz de riesgos

	Riesgo	Probabilidad	Impacto	Respuesta	Planes
NEGATIVOS	No tener un espacio de trabajo adecuado	Impepinable	Medio	Mitigar	Definir horarios de trabajo Adecuar el espacio de trabajo que se tiene
	Daños en los equipos de trabajo	Baja	Crítico	Transferir	Solicitar a Icesi el préstamo de equipos de cómputo Solicitar cambio por garantía
	La curva de aprendizaje para el uso de nuevas tecnologías es más larga de lo esperado	Media	Medio	Mitigar	Solicitar ayuda a una persona experta en el tema
	Cambios sintácticos o semánticos en HLVL	Baja	Medio	Aceptar	Trabajar con la misma versión de HLVL
	Los ciclos de revisión por parte de los tutores son más lentos de lo esperado	Alta	Alto	Mitigar	Definir un cronograma de trabajo donde se establezcan entregas con tiempo suficiente, teniendo en cuenta el horario de los profesores
	Posibilidad de que alguno de los integrantes no pueda matricularse el siguiente semestre, dada las condiciones que presenta el COVID-19	Media	Crítico	Aceptar	Ajustar el alcance del proyecto
POSITIVOS	Terminar el proyecto antes del tiempo establecido	Media	Crítico	Aceptar	Aprovechar el tiempo de vacaciones en cuarentena para adelantar el desarrollo de las primeras fases del proyecto
	Contribuir con la extensión de HLVL puede dar lugar a una publicación	Baja	Baja	Aceptar	Ser autores de una publicación de divulgación del proyecto

Referencias Bibliográficas

- Abele, A., Papadopoulos, Y., Servat, D., Törngren, M., Weber, M., Johansson, R., Lönn, H., Papadopoulos, Y., Reiser, M.-O., Servat, D., Törngren, M., & Weber, M. (2010). The CVM Framework - A Prototype Tool for Compositional Variability Management. *VAMOS'10: Proceedings of the 4th International Workshop on Variability Modelling of Software-Intensive Systems*, 1(October 2014), 101–105.
- Acher, M. (2011). Managing Multiple Feature Models: Foundations , Language and Applications. *Sophia*, 51–53. <http://www.mathieuacher.com/PhDAcher2011-revised.pdf>
- Acher, M., Collet, P., Lahire, P., & France, R. (2010a). Comparing Approaches to Implement Feature Model Composition. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*: Vol. 6138 LNCS (pp. 3–19). https://doi.org/10.1007/978-3-642-13595-8_3
- Acher, M., Collet, P., Lahire, P., & France, R. (2010b). Composing feature models. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5969 LNCS, 62–81. https://doi.org/10.1007/978-3-642-12107-4_6
- Acher, M., Collet, P., Lahire, P., & France, R. B. (2013). FAMILIAR: A domain-specific language for large scale management of feature models. *Science of Computer Programming*, 78(6), 657–681. <https://doi.org/10.1016/j.scico.2012.12.004>
- Anderson, J. S., & Fickas, S. (1989). *Proposed perspective shift: viewing specification design as a planning problem*. 177–184. <https://doi.org/10.1145/75200.75227>
- Asadi, M., Soltani, S., Gasevic, D., Hatala, M., & Bagheri, E. (2014). Toward automated feature model configuration with optimizing non-functional requirements. *Information and Software Technology*, 56(9), 1144–1165. <https://doi.org/10.1016/j.infsof.2014.03.005>
- Bachmann, F., & Clements, P. C. (2005). Variability in software product lines. Technical Report CMU/SEI-2005-TR012. *Software Engineering Institute, Pittsburgh, USA, September*, 46.
- Benavides, D., Segura, S., & Ruiz-Cortés, A. (2010). Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6), 615–636. <https://doi.org/10.1016/j.is.2010.01.001>
- Berger, T., Rublack, R., Nair, D., Atlee, J. M., Becker, M., Czarnecki, K., & Wąsowski, A. (2013). A survey of variability modeling in industrial practice. *ACM International Conference*

Proceeding Series, 8. <https://doi.org/10.1145/2430502.2430513>

- Chavarriaga, J. (2017). *Using multiple Feature Models of Domains and Regulations to develop Configuration Systems*. February, 215.
- Clarke, D., Muschevici, R., Proença, J., Schaefer, I., & Schlatte, R. (2011). Variability Modelling in the ABS Language. In *Lecture Notes in Computer Science* (Vol. 3590, pp. 204–224). https://doi.org/10.1007/978-3-642-25271-6_11
- Classen, A., Boucher, Q., & Heymans, P. (2011). A text-based approach to feature modelling: Syntax and semantics of TVL. *Science of Computer Programming*, 76(12), 1130–1143. <https://doi.org/10.1016/j.scico.2010.10.005>
- Czarnecki, K., Helsen, S., & Eisenecker, U. (2004). Staged configuration using feature models. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3154, 266–283. https://doi.org/10.1007/978-3-540-28630-1_17
- Czarnecki, K., & Wasowski, A. (2008). *Feature Diagrams and Logics: There and Back Again*. January 2015, 23–34. <https://doi.org/10.1109/spline.2007.24>
- Eichelberger, H., & Schmid, K. (2015). Mapping the design-space of textual variability modeling languages: a refined analysis. *International Journal on Software Tools for Technology Transfer*, 17(5), 559–584. <https://doi.org/10.1007/s10009-014-0362-x>
- El-Sharkawy, S., Kröher, C., & Schmid, K. (2011). Supporting heterogeneous compositional multi software product lines. *ACM International Conference Proceeding Series*. <https://doi.org/10.1145/2019136.2019164>
- Galindo, J. A., Benavides, D., Trinidad, P., Gutiérrez-Fernández, A. M., & Ruiz-Cortés, A. (2019). Automated analysis of feature models: Qo vadis? *ACM International Conference Proceeding Series*, A. <https://doi.org/10.1145/3336294.3342373>
- Hevner, A. R. (2004). *A Three Cycle View of Design Science Research A Three Cycle View of Design Science Research*. 19(2), 87–92. <https://www.researchgate.net/publication/254804390>
- K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, e A. S. P. (1990). *Feature-Oriented Domain Analysis(FODA) Feasibility Study - A Technical Report* (Vol. 98). Institute of Clinical Research. <http://www.dtic.mil/docs/citations/ADA235785>
- Krueger, C. W. (1992). Software reuse. *ACM Computing Surveys (CSUR)*, 24(2), 131–183. <https://doi.org/10.1145/130844.130856>
- Lopez-Herrejon, R. E., & Batory, D. (2001). A standard problem for evaluating product-line methodologies. *Lecture Notes in Computer Science (Including Subseries Lecture Notes*

- in Artificial Intelligence and Lecture Notes in Bioinformatics*), 2186(Gcse), 10–24.
https://doi.org/10.1007/3-540-44800-4_2
- Peffers, K., Tuunanen, T., & Gengler, C. (2006). The design science research process: a model for producing and presenting information systems research. *Journal of Management Information Systems*, February, 83–106.
- Pohl, K., Böckle, G., & van der Linden, F. (2005). Software Product Line Engineering. In *Communications of the ACM* (Vol. 49, Issue 12). Springer Berlin Heidelberg.
<https://doi.org/10.1007/3-540-28901-1>
- Rosenmüller, M., Siegmund, N., Thüm, T., & Saake, G. (2011a). Multi-dimensional variability modeling. *ACM International Conference Proceeding Series, May 2014*, 11–20.
<https://doi.org/10.1145/1944892.1944894>
- Rosenmüller, M., Siegmund, N., Thüm, T., & Saake, G. (2011b). Multi-dimensional variability modeling. *ACM International Conference Proceeding Series*, 11–20.
<https://doi.org/10.1145/1944892.1944894>
- Schmid, K. (2010). Variability modeling for distributed development - A comparison with established practice. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6287 LNCS, 151–165. https://doi.org/10.1007/978-3-642-15579-6_11
- Schmid, K., Kröher, C., & El-Sharkawy, S. (2015). *Integrated Variability Modeling Language: Language Specification* (Issue lvml).
- Ter Beek, M. H., Schmid, K., & Eichelberger, H. (2019). Textual variability modeling languages an overview and considerations. *ACM International Conference Proceeding Series, B*. <https://doi.org/10.1145/3307630.3342398>
- Villota, A., Mazo, R., & Salinesi, C. (2019). The high-level variability language: An ontological approach. *ACM International Conference Proceeding Series, B*. <https://doi.org/10.1145/3307630.3342401>
- Fundación Wikipedia Inc. (7 de October de 2019). *Wikipedia*. Obtenido de Wikipedia:
<https://en.wikipedia.org/wiki/Solver>
- Fundación Wikipedia Inc. (3 de Abril de 2020). *Wikipedia*. Obtenido de Wikipedia:
<https://es.wikipedia.org/wiki/GPS>
- Fundación Wikipedia Inc. (9 de Abril de 2020). *Wikipedia*. Obtenido de Wikipedia:
<https://es.wikipedia.org/wiki/MP3>

Fundación Wikipedia Inc. (8 de Marzo de 2020). *Wikipedia*. Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Ad_hoc

Fundation Wikipedia Inc. (28 de Marzo de 2020). *Project stakeholder: Fundation Wikipedia Inc.* Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Project_stakeholder

Howson, I. (2 de Mayo de 2019). *rdr.io*. Obtenido de <https://rdr.io/cran/fame/>

www.cs.utexas.edu. 2020. SATLINK - Dimacs. [online] Available at: <https://www.cs.utexas.edu/users/moore/acl2/manuals/current/manual/index-seo.php/SATLINK___DIMACS> [Accessed 2 December 2020].