Análisis de Modelos de Variabilidad Especificados en Kconfig

Matías Sequeira
Universidad Tecnológica Nacional - Facultad Regional Santa Fe
Lavaise 610, Santa Fe, Argentina
matiassequeirq@googlemail.com

Rocío González
Universidad Tecnológica Nacional - Facultad Regional Santa Fe
Lavaise 610, Santa Fe, Argentina
rociogonzalez@outlook.es

Silvio Gonnet
INGAR, (Conicet - UTN)
Universidad Tecnológica Nacional - Facultad Regional Santa Fe
Avellaneda 3657, Santa Fe, Argentina
sgonnet@santafe-conicet.gov.ar

Abstract

La industria de software ha comenzado a centrar el desarrollo de sus productos siguiendo un enfoque de líneas de productos de software. En este tipo de desarrollo la industria del software requiere de herramientas que permitan especificar la variabilidad de sus productos. En la actualidad, existe una marcada tendencia al empleo del lenguaje Kconfig para expresar tal variabilidad. Asimismo, el alto nivel de variabilidad y las interdependencias entre características posibles en los distintos productos requieren de herramientas que brinden soporte en el análisis de la variabilidad, identificando posibles inconsistencias. En la literatura existen distintas propuestas para realizar tales análisis. Sin embargo, estas alternativas están basadas en modelos de características, no incluyendo a modelos expresados en Kconfig. A partir del escenario planteado, en este trabajo se propone la construcción de una herramienta de software que traduce un modelo de variabilidad especificado en Kconfig a un modelo de características. Este modelo traducido es analizado mediante el soporte de herramientas existentes para el análisis de la variabilidad de modelos de características.

Palabras Clave
Kconfig; variabilidad; característica.

1. Introduction

Una tendencia creciente en el desarrollo de software es la necesidad de desarrollar múltiples productos de software similares en conjunto con un alto grado de flexibilidad en vez de varios productos individuales [1-2]. Hay varias razones para esto: los productos están enfocados a distintos sectores del mercado, sujetos a distintas restricciones legales o culturales, o deben satisfacer necesidades específicas de diferentes "stakeholders". Estas exigencias del mercado actual, en cuanto a flexibilidad y adaptabilidad, se pueden satisfacer a partir del desarrollo de software altamente configurable o mediante líneas de producto de software. En estos tipos de arquitecturas de desarrollo de software las posibilidades de configuración, y adaptación, se expresan por medio de la variabilidad [3]. Se conoce como variabilidad a la posibilidad que posee un sistema de ser extendido, cambiado, localizado, o configurado para su uso en un contexto particular [4].

Una línea de productos de software está constituida por un núcleo que contiene los componentes presentes en todos los productos o aplicaciones derivadas, y un conjunto de elementos variables, "variabilidad", que incluye aquellas características optativas de las aplicaciones [2, 5]. La definición de la variabilidad es una de las actividades centrales de la ingeniería de dominio de una línea de productos de software; en ella se establecen las variantes, dependencias y las relaciones con los artefactos de software (casos de uso, arquitectura, código, etc.). Por otra parte, la ingeniería de aplicaciones se encarga de derivar los productos finales a partir del núcleo principal y las distintas variantes posibles [2].

Los conceptos previamente introducidos en el contexto de línea de productos de software también se han empleado en la industria del software para la representación de software/sistemas altamente configurables. Los sistemas altamente configurables proveen opciones de configuración, también conocidas como características para adaptar el sistema según un conjunto de requerimientos [6]. Una opción de

configuración representa una opción de inclusión de una cierta funcionalidad o de un requerimiento de calidad en una variante del sistema. No todas las opciones de configuración son permitidas o válidas; además, existen restricciones y dependencias entre ellas. Estas opciones de configuración y las restricciones permiten definir la variabilidad del producto.

Metzger y Pohl [7] enfatizan que para gestionar cuestiones de variabilidad es indispensable documentar de manera explícita este aspecto. En consecuencia, la variabilidad se especifica en lo que se conoce como modelo de variabilidad [6-9], el cual, dependiendo el enfoque utilizado puede ser un modelo de características [10], un modelo de decisión [11], o un modelo expresado en CVL [12]. Asimismo, existen otras representaciones para sistemas altamente configurables, como el lenguaje Kconfig [13], el cual es empleado en sistemas "open-source" [14-15], principalmente en el kernel de Linux. Estos modelos de variabilidad describen y organizan las características comunes y variables de productos [14], y son las entradas a las herramientas de configuración, que asisten al usuario en resolver variabilidades y derivar un producto concreto [8].

Un sistema altamente configurable puede llegar a tener un gran número de características. Reportes de sistemas industriales indican poseer cientos de características [16] y en sistemas "open-source", la cantidad de características asciende a miles [14]. En consecuencia, un producto altamente configurable puede contener un número excesivamente grande de productos individuales [14, 17]. En el proceso de definición de la variabilidad es posible introducir inconsistencias y ambigüedades, lo que podría resultar en modelos confusos y contradictorios que dificultan tanto la derivación de nuevas aplicaciones, como el mantenimiento de aplicaciones existentes. Las características en un sistema configurable interactúan de una manera no trivial, y su interacción puede introducir errores en los productos derivados [18]. Consecuentemente, es necesario contar con métodos para razonar sobre los posibles productos de una manera eficiente, al igual que brindar soporte para las operaciones identificadas en Kang y colab. [10] y Benavides y colab. [19], tales como: a) detección de inconsistencias, b) detección de características muertas (no pueden incluirse en ningún modelo), c) número de productos derivables. Debido al incremento en tamaño y complejidad de los modelos, surge el desafío de proporcionar un soporte automático para llevar adelante estas funciones. Muchas de estas funcionalidades están disponibles en el ambiente SPLOT, www.splot-research.org [20], una herramienta académica orientada a la definición de modelos de características y su análisis mediante el empleo de lógica proposicional y problemas de satisfacción de restricciones. Sin embargo, los ejemplos disponibles son casos de estudios pequeños y no reflejan modelos del mundo real como los expresados mediante Kconfig. Diversos autores [21-22] indican que un modelo de variabilidad especificado en Kconfig puede ser interpretado como un modelo de características.

A partir del escenario planteado, en este trabajo se propone una herramienta para el análisis de la variabilidad especificada en Kconfig, la propuesta se basa en la traducción de un modelo de variabilidad especificado en Kconfig a un modelo de características. Dicho modelo de características luego es analizado para poder finalmente obtener información de su variabilidad.

En la siguiente sección se introducen los modelos de variabilidad empleados en el trabajo. Luego, en la sección 3 se incluye la propuesta de traducción del lenguaje Kconfig a modelos de características. Luego, en la sección Resultados y Discusión se explican las funcionalidades obtenidas con el uso de una herramienta que implementa la propuesta. Asimismo, se presentan las fortalezas y debilidades de la herramienta desarrollada. Por último, se presentan las conclusiones desde diferentes puntos de vista y la propuesta para trabajos futuros.

2. Modelos de Variabilidad

La variabilidad se define como la habilidad de cambio o de personalización de un sistema [4], esta variabilidad es representada mediante un modelo de variabilidad. Sin embargo, no existe un único lenguaje para representar la variabilidad. Existen dos grandes grupos en los que se dividen los sistemas con variabilidad: los sistemas personalizables, donde la variabilidad se debe principalmente a la selección realizada por el usuario de las partes que más le interesa; y las familias de productos [23], donde una serie de productos medianamente similares se unen para permitir la reutilización de la parte común. La principal diferencia con los sistemas tradicionales es que se debe prestar un especial interés al análisis de la parte común y de las partes variables, estableciendo las dependencias entre ellas. En la literatura se reportan diversas operaciones para realizar tal análisis, la mayoría de las propuestas están basadas en modelos de variabilidad representados mediante modelos de características. Estos resultados no son directamente aplicables a los sistemas altamente configurables, debido a que su variabilidad es representada mediante lenguajes como Kconfig. En consecuencia, en esta sección se presenta la representación de la variabilidad mediante los lenguajes Kconfig y modelos de características.

2.1. Representación de la Variabilidad empleando el Lenguaje Kconfig

Kconfig [13] fue creado para describir la variabilidad del kernel de Linux y ha sido adoptado por diversos

proyectos de desarrollo de código abierto para definir su variabilidad [14, 21].

En Kconfig una opción de configuración, característica, es representada mediante el constructor "config". El lenguaje Kconfig permite definir las distintas opciones de configuración y las dependencias entre las mismas. En este lenguaje las configs pueden estar anidadas dentro de otras opciones de configuración empleando el constructor "menuconfig"; pueden estar agrupadas en menús (constructor "menu"), en grupos de opciones de configuraciones (mediante el constructor "choice") o en bloques opcionales (empleando el constructor "if"). Estos agrupadores pueden anidarse entre sí de manera indistinta excepto los grupos del tipo choice, los cuáles sólo pueden contener configs. La herramienta xconfig permite al usuario seleccionar el conjunto de opciones de configuraciones con las cuales construirá el producto final, para el caso de Linux el producto final es su kernel. Esta herramienta presenta el modelo Kconfig como un árbol de opciones, donde el usuario selecciona las configuraciones que desea para construir el producto final.

Cada opción de configuración (config) posee un nombre y un tipo. Los tipos pueden ser: "bool", "tristate", "integer" ("int" o "hex"), o "string" [13, 21]. Una opción de configuración de tipo boolean representa una opción que puede ser seleccionada como parte del producto final ("y") o no ("n"). Una config del tipo tristate es similar a la del tipo boolean, pero posee dos alternativas de inclusión en el producto final: "y" indica que el código que implementa la opción es enlazado en el kernel de manera estática, mientras que "m" representa que debe ser compilado como un módulo que carga de manera dinámica. Las opciones de tipo integer se emplean para especificar opciones numéricas, tal como el tamaño de un buffer. Las opciones de tipo string permiten especificar el nombre de un elemento configurable, como puede ser el nombre de una partición del disco. She et al. [21] denominan a las configs de tipo boolean y tristate como "switch configs", en cambio a las configs de tipo integer y string la denominan "entry-field config".

Estas configs pueden estar agrupadas mediante menús. Además, es posible agrupar configs en un bloque if bajo una determinada expresión. Esta expresión depende de otras configs o menuconfigs. Sólo es posible configurar los elementos que están dentro del bloque if cuando la expresión del bloque evalúa verdadera. Por esto, se podría pensar que la especificación if agrega una dependencia entre los componentes del bloque y las opciones de configuraciones que participan en la expresión que evalúa.

En la Figura 1 se incluye un ejemplo de un menuconfig (W1) seguido de la sentencia if, lo cual representa el comienzo de un bloque if. A su vez, el bloque if contiene una config (W1_CON) y un menu (1-wire Bus Masters) compuesto por tres configs de tipo Tristate. Estos elementos dependen del valor de la config W1 y sólo podrán ser configurados si W1 es seleccionado (valores "y" o "m").

Kconfig provee un mecanismo de visibilidad condicional para los elementos del lenguaje, colocando una especificación "depends on" debajo de los mismos. Una especificación depends on introduce una dependencia que debe ser satisfecha cuando se selecciona la config. Si la condición es falsa, el elemento en cuestión y sus hijos son descartados por el configurador. En la Figura 1, W1_MASTER_MXC depende de la selección de al menos una de las opciones: MMU o SBUS.

Inverso a lo anteriormente explicado, una especificación "select" obliga la selección de otra config cuando la config es seleccionada por el usuario. En el caso representado por la Figura 1, cuando se selecciona W1_MASTER_DS2482 se debe seleccionar el elemento X86_HT.

```
config X86_HT
        bool
config MMU
        def_bool y
config SBUS
        bool
menuconfig W1
        tristate "Dallas's 1-wire support"
if W1
config W1_CON
        bool "Userspace communication over connector"
        default y
menu "1-wire Bus Masters"
config W1_MASTER_DS2490
        tristate "DS2490 USB <-> W1 transport layer"
config W1_MASTER_DS2482
        tristate "Maxim DS2482 I2C to 1-Wire bridge"
        select X86_HT
config W1_MASTER_MXC
        tristate "Freescale MXC 1-wire busmaster"
        depends on MMU || SBUS
endmenu
endif
```

Figura 1. Modelo parcial de variabilidad del kernel de Linux expresado en Kconfig.

Los grupos de opciones (choice configs) permiten definir alternativas. Las choice configs heredan el tipo de las configs contenidas dentro del grupo. Las opciones pueden ser bool o tristate. Cuando la choice adquiere el tipo bool se debe seleccionar una única opción (XOR) [1...1], como si la misma estuviera seleccionada ("y"); cuando adquiere el tipo tristate, se puede seleccionar una o más opciones (OR) [1…*], como si hubiera adquirido el valor "m". La choice de la Figura 2 es un ejemplo de XOR y al momento de configurar un producto se deberá seleccionar una única opción de los configs que contiene: THERMAL_DEFAULT_GOV_STEP_WISE, o

THERMAL_DEFAULT_GOV_FAIR_SHARE.
Adicionalmente, un grupo de opción marcado como opcional ("optional") puede ser configurada con el valor "n", es decir, no es necesario seleccionar algunas de sus opciones ([0...1] para bool y [0…*] para tristate). Sin embargo, un grupo de opciones sin una marca de opcional es considerado obligatorio.

```
menuconfig THERMAL
        tristate "Generic Thermal sysfs driver"
if THERMAL
config THERMAL_HWMON
        bool
        prompt "Expose thermal sensors as hwmon
device"
        default y
choice
        prompt "Default Thermal governor"
config THERMAL_DEFAULT_GOV_STEP_WISE
        bool "step_wise"
        select THERMAL_HWMON
config THERMAL_DEFAULT_GOV_FAIR_SHARE
        bool "fair_share"
endchoice
endif
```

Figura 2. Modelo parcial de variabilidad del kernel de Linux expresado en Kconfig: uso de menuconfig y choice.

El modelo de la versión 4.2 del kernel Linux posee más de 10000 cláusulas config, 200 menuconfig, 170 menu, 60 choice, 6900 definiciones de select, y 9000 depends on. Al momento de la redacción de este trabajo, la última versión del kernel es la 4.12.8. Es de esperarse un modelo aún más grande debido al continuo crecimiento de este sistema operativo.

2.2. Representación de la Variabilidad empleando un Modelo de Características (Feature Model)

Un modelo de características es una representación compacta de todos los productos de una línea de productos de software en términos de características ("features"). Una "feature" es definida como un aspecto prominente o distintivo visible por el usuario, calidad, o característica de un sistema de software o sistema [10].

Las características se representan mediante una estructura de árbol, en donde se vinculan características padres con sus hijos. Además, pueden existir restricciones que afecten a dos o más características de cualquier lugar del modelo. Las relaciones permitidas entre características son las siguientes:

- **Obligatoria**: relación que vincula una característica hija con su padre, indicando que la hija aparecerá en todos los productos en los que el padre esté incluido.
- **Opcional**: relación que vincula una característica hija con su padre, indicando que la hija podrá opcionalmente estar incluida en algunos productos en los que esté incluido su padre.
- **Cardinalidad grupal**: la cardinalidad grupal relaciona al padre con un grupo de características hijas especificando el intervalo [min..máx]. Este intervalo limita el número de características hijas que pueden ser incluidas en un producto, cuando su padre está incluido, siendo min .. max, los límites inferior y superior, respectivamente.
- **Requiere**: esta relación indica una implicación entre una característica restringida y una característica requerida. La característica restringida sólo puede incluirse en aquellos productos en los que se encuentre la característica requerida.
- **Excluye**: esta relación indica una exclusión mutua entre dos características.

En la Figura 3 se ilustra un ejemplo de un modelo de característica pequeño, el cual es ilustrado empleando la herramienta SPLOT.

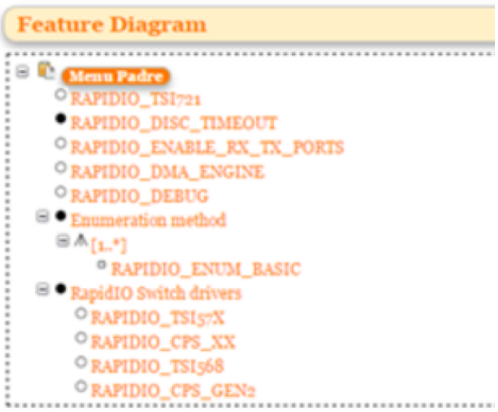

Figura 3. Ejemplo de diagrama de características: arquitectura Rapid IO.

Este modelo representa una porción de la variabilidad del kernel de Linux, en particular se ilustra la arquitectura Rapid IO. Los círculos negros de la Figura 3 indican una relación de obligatoriedad entre las características padre e hijas. Los círculos blancos representan una relación de tipo opcional entre padre e hijo. Las relaciones de cardinalidad grupal se indican con sus valores mínimos y máximos debajo de la característica padre. Según el modelo de la Figura 3, toda arquitectura Rapid IO (representado por

Menu Padre en Figura 3) debe contar con las propiedades RAPID_DISC_TIMEOUT, un Enumeration method, y un RapidIO Switch drivers. Opcionalmente puede incluir las características: RAPIDIO_TSI721, RAPIDIO_ENABLE_RX_TX_PORTS, RAPIDIO_DMA_ENGINE, y RAPIDIO_DEBUG.

3. Herramienta de análisis de modelos de variabilidad especificados en Kconfig

En este trabajo se aborda el análisis de la variabilidad en modelos especificados con el lenguaje Kconfig empleando herramientas de análisis de variabilidad de modelos de características. Para poder realizar tal análisis es necesario traducir un modelo expresado en Kconfig a un modelo de característica.

Para la construcción de la herramienta se propuso una arquitectura de tipo Pipe & Filter (Figura 4). El primer componente es un Parser cuya entrada es el modelo Kconfig y es el encargado de generar las instancias del modelo conceptual de Kconfig. Luego, el Transformador aplica un conjunto de reglas para traducir las instancias del modelo de Kconfig en instancias del modelo de características. Este modelo de características es especificado empleando la notación de la herramienta SPLOT. Por último, el analizador aplica las funcionalidades obtenidas a partir de la biblioteca de la herramienta SPLOT, la cual está basada en un SAT-solver, para realizar el análisis de variabilidad y posteriormente entregar la información requerida. El analizador permite:

- Reconocer inconsistencias: las inconsistencias se dan cuando existe una relación entre características que no pueden ser verdaderas al mismo tiempo.
- Determinar características muertas: las características muertas son las opciones de configuración que no aparecen en ningún producto derivable debido a un conjunto de restricciones.
- Determinar características opcionales: las características opcionales definen una alternativa para el producto base que puede ser seleccionada como parte de un producto final o no.

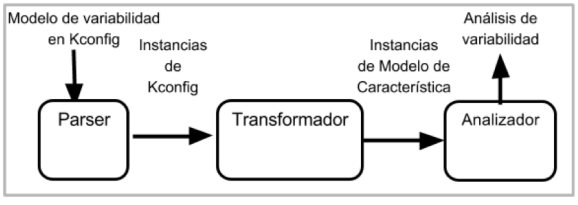

Figura 4. Arquitectura de la herramienta.

- Validar producto ingresado: dado un conjunto de configuraciones, se debe poder decidir si las mismas conforman un producto válido. Se considera válido un producto cuando no contiene inconsistencias ni opciones de configuraciones obligatorias faltantes.
- Obtener cantidad de productos derivables.

3.1. Reglas para la traducción de un modelo Kconfig a un modelo de características

A continuación, se explicarán las reglas definidas para el mapeo de conceptos de Kconfig a conceptos de modelos de características.

Todo modelo de característica es representado mediante un árbol, cuya raíz se la denomina "_r" y como hijos contiene a los elementos del archivo Kconfig.

Aquellas opciones de configuración definidas mediante config del tipo bool o tristate son traducidas como características opcionales, representado como ":o" en el modelo de característica en SPLOT. El resto de las opciones de configuración son traducidas como características obligatorias, indicadas como ":m" en SPLOT. En la Tabla 1 se provee un breve ejemplo de la representación de la variabilidad en una bicicleta mediante el lenguaje Kconfig y su traducción a modelo de característica. En la parte superior se incluye la definición de dos opciones de configuración: SHIFT y BRAND. La primera de ellas es del tipo bool y la segunda de tipo string. En la parte inferior de la Tabla 1 se incluye la traducción a modelo de característica donde se especifica que la opción cambios (SHIFT) son características opcionales (:o SHIFT) y es obligatorio que el producto posea una marca, la config BRAND se traduce en :m BRAND.

Tabla 1. Traducción de config.

Modelo expresado en Kconfig

config SHIFT
  bool "21 speed shift"
config BRAND
  string "Bike brand"

Modelo de característica (expresado en SPLOT)
:o SHIFT (_r_5) :m BRAND (_r_6)

Los elementos contenidos dentro de un menú (config, menuconfig, if, choice, y otros menús), pasan a ser hijo del menú. En la Tabla 2 se muestra un menú, el cual agrupa las opciones de configuraciones opcionales para la inclusión de frenos a discos en cada rueda.

En las Tablas 1, 2, y 3 se puede observar el símbolo r_N_N..., esta notación sirve para identificar de manera unívoca las características dentro del árbol de características: por cada nivel dentro del árbol de características se incorpora un elemento _N al

identificador y su prefijo es igual al identificador del padre. Por ejemplo, en la Tabla 2, suponiendo que el menú fue traducido como la característica "_r_5", las características que representan las configs hijas del menú serán "_r_5_6" y "_r_5_7". La selección del próximo número a utilizar sigue un criterio ascendente.

Tabla 2. Traducción de menu.

Modelo expresado en Kconfig

menu "Disc brake system"
config REAR_DISC
  bool "Rear wheel disc brake"
config FRONT_DISC
  bool "Front wheel disc brake"
endmenu

Modelo de característica (expresado en SPLOT)

:m "Disc brake system" (_r_5)
  :o REAR_DISC (_r_5_6)
  :o FRONT_DISC (_r_5_7)

Una opción de configuración choice es traducida a cardinalidad grupal según el "type" de las opciones configs que la componen: a) si son del tipo "bool", sólo puede seleccionarse una de las opciones, por lo que se la traduce a [1..1]; b) en caso de ser del tipo "tristate", se pueden seleccionar uno o más componentes con el valor "m", siendo entonces traducida a [1..*]. Cuando la choice posee "optional" dentro de sus atributos se permite la no selección de componentes, entonces, la cardinalidad resulta ser [0..1] o [0..*], respectivamente. En la Tabla 3 se muestra un ejemplo de configuración del cuadro (frame) de una bicicleta, con una choice que exige seleccionar un único ([1..1]) tipo de cuadro: de aluminio o de carbón. La cardinalidad grupal se representa en SPLOT mediante la notación :g [min, max] seguida de las distintas opciones :nombreDeLaOpción.

Tabla 3. Traducción de choice.

Modelo expresado en Kconfig

choice
  prompt "Frame"
config FRAME_ALUMINIUM
  bool
config FRAME_CARBON
  bool
endchoice

Modelo de característica (expresado en SPLOT)

:m "Frame" (_r_5)
  :g [1,1]
  :FRAME_ALUMINIUM (_r_5_6)
  :FRAME_CARBON (_r_5_7)

Los menuconfigs son traducidos de forma similar a las opciones configs. Generalmente, cuando se encuentra un menuconfig, debajo del mismo aparece un bloque "if" cuya condición a evaluar es la config del menuconfig. Esto se puede pensar como que si todo lo que está dentro del bloque "if" es hijo del menuconfig. En caso de no estar precedido por un bloque "if", el menuconfig es tratado como una config. En la Tabla 4 se brinda un ejemplo de la opción accesorios (ACCESORIES) cuya inclusión es opcional al momento de configurar un producto correspondiente a una bicicleta. Dentro de los posibles accesorios se incluyen las opciones luz de emergencia (EMERGENCY_LIGHT) y espejo (MIRROR).

Tabla 4. Traducción de menuconfig.

Modelo expresado en Kconfig

menuconfig ACCESORIES
  bool "Not mandatory parts"
if ACCESORIES
  config EMERGENCY_LIGHT
  bool
  config MIRROR
  bool
endif

Modelo de característica (expresado en SPLOT)

:o ACCESORIES (_r_5)
  :o EMERGENCY_LIGHT (_r_5_6)
  :o MIRROR (_r_5_7)

Adicionalmente, cuando la expresión a evaluar para un bloque if se encuentra compuesta por un símbolo negado o por más de un símbolo unidos por conectores lógicos (por ejemplo !A o A || B respectivamente), el contenido del bloque if se representa mediante restricciones del tipo "depends on" donde el contenido del bloque if depende de la expresión del mismo, ya que no puede ser representado mediante la estructura de árbol.

El árbol de característica es completado con un conjunto de restricciones para representar las relaciones de tipo requiere y excluye.

Las restricciones son traducidas por la herramienta en fórmulas proposicionales en forma normal conjuntiva (FNC), cada cláusula representa una restricción.

La restricción del tipo select es representada por una implicancia unidireccional, la cual es expresada con una disyunción, es decir, si A select B, entonces se tiene que $A \Rightarrow B$. En FNC se representa como ~A or B. Se puede observar un ejemplo en la Tabla 5, en este caso, si se selecciona PEDAL_ALUMINIUM (_r_8) se deberá seleccionar el cuadro FRAME_ALUMINIUM (_r_5_6). Esto es traducido como la restricción ~_r_8 or _r_5_6.

Cuando una entrada posee la opción "depends on" y la expresión que le sigue a dicha opción hace referencia a sólo una entrada (el símbolo de un menú, una config, etc.), se traduce como una restricción del tipo requiere, donde la entrada que posee el depends on implica la entrada referenciada por el depends on. En la Tabla 6 se provee un ejemplo.

Tabla 5. Traducción de select.

Modelo expresado en Kconfig

choice
  prompt "Frame material"
config FRAME_ALUMINIUM
  bool
config FRAME_CARBON
  bool
endchoice
config PEDAL_ALUMINIUM
  bool
  select FRAME_ALUMINIUM

Modelo de característica (expresado en SPLOT)

En el árbol:
:m "Frame material" (_r_5)
  :g[1,1]
  :o FRAME_ALUMINIUM (_r_5_6)
  :o FRAME_CARBON (_r_5_7)
:o PEDAL_ALUMINIUM (_r_8)
 	
En las restricciones:
constraint_N: ~_r_8 or _r_5_6

Tabla 6. Traducción de depends on.

Modelo expresado en Kconfig

config RACE_RIMS
  bool "Light-weight, double wall rims"
config RACE_TIRES
  bool "Thin tires"
  depends on RACE_RIMS

Modelo de característica (expresado en SPLOT)
En el árbol: :o RACE_RIMS (_r_5) :o RACE_TIRES (_r_6) En las restricciones: constraint_N: ~_r_6 or _r_5

Por lo contrario, si la expresión del depends on hace referencia a más de una entrada, la restricción es transformada como si-sólo-si. Por ejemplo, si A depends on B, siendo B una expresión que involucra más de un elemento, se representa como A $\Leftrightarrow$B. Internamente son manejadas como dos restricciones: A $\Rightarrow$ B y B $\Rightarrow$ A. En la Tabla 7 se incluye un caso para B = X or Y, y en la Tabla 8 se especifica cuando B = X and Y. B puede poseer dos o más elementos, para la transformación se obtiene la FNC, siendo cada cláusula una restricción.

Tabla 7. Traducción de depends on X || Y.

Modelo expresado en Kconfig
config A   depends on X II Y
Modelo de característica (expresado en SPLOT)
En el árbol: :o/m A (_r_5) :o/m X (_r_6) :o/m Y (_r_7) En las restricciones: constraint_N1: ~_r_5 or _r_6 or _r_7 constraint_N2: ~_r_6 or _r_5 constraint_N3: ~_r_7 or _r_5

Tabla 8. Traducción de depends on X && Y.

Modelo expresado en Kconfig
config A   depends on X && Y
Modelo de característica (expresado en SPLOT)
En el árbol: :o/m A (_r_5) :o/m X (_r_6) :o/m Y (_r_7) En las restricciones: constraint_N1: ~_r_5 or _r_6 constraint_N2: ~_r_5 or _r_7 constraint_N3: ~_r_6 or ~_r_7 or _r_5

4. Resultados y Discusión

La herramienta propuesta fue implementada en Java, empleando ANTLR4 (http://www.antlr.org/) para la realización del parser y la biblioteca SPLAR para una porción de la funcionalidad requerida del análisis de la variabilidad (reconocimiento de características muertas, características opcionales y obtención de cantidad de productos derivables). La biblioteca SPLAR está basada en el solver SAT4j para el razonamiento de satisfacibilidad lógica (SAT), y es la biblioteca empleada para el razonamiento en la herramienta SPLOT (http://www.splot-research.org/). Para la implementación del resto de las funcionalidades de análisis de variabilidad

se tuvo que extender SPLAR y hacer un uso combinado de las funciones ofrecidas.

A continuación, se ilustran las capacidades de la herramienta, mediante una explicación de las funcionalidades y salidas de las mismas, para luego poder analizar los resultados alcanzados con la propuesta.

Al iniciar la herramienta, la pantalla de presentación es la mostrada en la Figura 5. Además, se puede observar el modelo Kconfig a analizar, el cual corresponde a una familia de productos de una bicicleta.

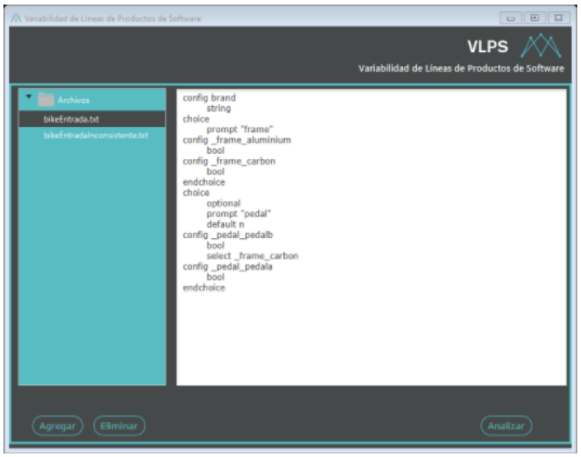

Figura 5. Pantalla inicial con modelo de variabilidad.

Al presionar el botón "Analizar" se transforma el modelo Kconfig a un modelo de características. Luego, al presionar el botón "Correr Análisis", se realiza un estudio sobre el modelo transformado, obteniéndose como resultado la validación de la LPS, las características comunes, las características opcionales, las características muertas y la cantidad de configuraciones. En la Figura 6, se muestra que para ese modelo de variabilidad se obtienen 3 características comunes, 5 opcionales y 5 configuraciones de productos posibles. Estas características son identificadas en el árbol que está en la izquierda de la ventana de la Figura 6. Las características comunes son Menu Padre (_r), brand (_r_6), y frame (_r_7). Particularmente, la característica Menu Padre (_r) es una característica ficticia utilizada para agrupar los componentes Kconfig que no tienen un padre, es decir, están en la raíz del archivo. Esta última característica estará presente en todos los modelos. Las características opcionales son _frame_aluminum (_r_7_8), _frame_carbon (_r_7_9), pedal (_r_10), _pedal_pedalb (_r_10_11) y _pedal_pedala (_r_10_12). Respecto de las configuraciones posibles, una de ellas es: Menu Padre (_r), brand (_r_6), frame (_r_7), _frame_carbon (_r_7_9), pedal (_r_10) y _pedal_pedalb (_r_10_11).

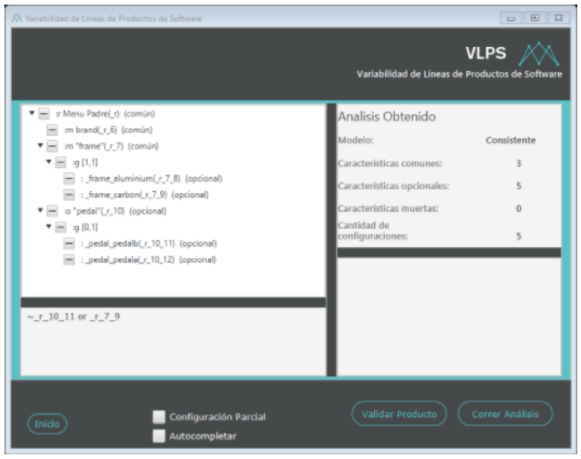

Figura 6. Modelo de características analizado.

En la Figura 7 se incluye una variante del modelo ilustrado anteriormente en la Figura 5. Esta modificación fue realizada para ilustrar la detección de "config _pedal_pedalb" como una característica muerta. Al modelo de la Figura 5 se le adiciona la relación "select" "_pedal_pedala" (_r_10_12) a la "config_frame_carbon" (_r_7_9), provocando que la "config_pedal_pedalb" (_r_10_11) no pueda aparecer en algún producto ya que, de ser seleccionada, implica la selección de "config_frame_carbon" y por transitividad, la selección de "config_pedal_pedala", violando así la restricción de grupo [0,1]. En la Figura 7 se puede ver el agregado de la restricción ~_r_7_9 or _r_10_12 como resultado de esta relación select, y la identificación de _r_10_11 como característica muerta.

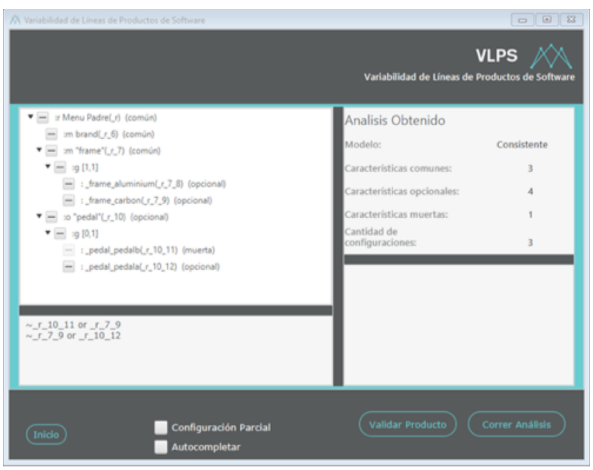

Figura 7. Identificación de característica muerta.

Además de indicar y contar la cantidad de características muertas, las mismas se deshabilitan para su

selección. En la Figura 7 se puede apreciar cómo se reduce la cantidad de configuraciones posibles (3) debido a la restricción adicional.

En la Figura 8 se incluye otra variante del modelo presentado en la Figura 5: se le adiciona la cláusula "select _frame_carbon" (_r_6_8) a la config "_frame_aluminium" (_r_6_7) y la cláusula "select _frame_aluminium" (_r_6_7) a la config "_frame_carbon" (_r_6_8). Estas dos relaciones select incorporan las restricciones ~_r_6_8 or _r_6_7 y ~_r_6_7 or _r_6_8 (parte inferior izquierda de la Figura 8).

Al presionar el botón "Correr Análisis", se obtiene que el modelo en cuestión es inconsistente, como se puede observar en la sección Análisis Obtenido de la Figura 8. Esto se debe a que la selección de la configuración del cuadro de aluminio implica la selección de la configuración del cuadro de carbono y viceversa, violando la restricción del grupo, la cual permite la selección de una y sólo una (:g [1,1]) configuración del tipo "frame" (cuadro).

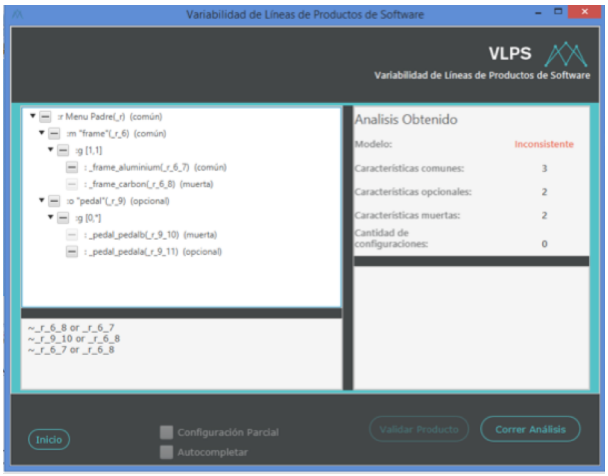

Figura 8. Modelo de características inconsistente.

En la Figura 9 se muestra la funcionalidad correspondiente a la validación de un producto ingresado. Se le permite al usuario seleccionar las características que desea incluir para la configuración de un producto mediante la selección de los "checkboxs" asociados a cada característica. Para la Figura 9, se ha decidido incluir sólo una característica, brand (_r_6), la cual es común. Luego de validar el producto, la herramienta ha detectado que las características frame (_r_7) y _frame_aluminium (_r_7_8) o frame_carbon (_r_7_9) deben ser incluidas de manera obligatoria. La herramienta también detecta cuando hay características seleccionadas que no pueden ser incluidas.

Un punto muy importante para destacar es la capacidad de procesar modelos de gran tamaño: se ha probado el análisis de modelos Kconfig de aproximadamente 3000 líneas, aunque con una demora considerable según la complejidad del modelo.

La problemática encontrada a partir de la traducción de un modelo de variabilidad a un modelo de características, es decir, traducción de un modelo expresado en Kconfig a la notación utilizada por SPLOT, se basa en la pérdida de expresividad del modelo de variabilidad en Kconfig. Esto se debe a la imposibilidad de incluir muchos atributos de las opciones de configuración config, los cuales pueden hacer que las mismas interactúen de manera distinta. Sin embargo, dada la complejidad del lenguaje Kconfig, se permite analizar un gran número de problemas que pueden presentarse a nivel estructural.

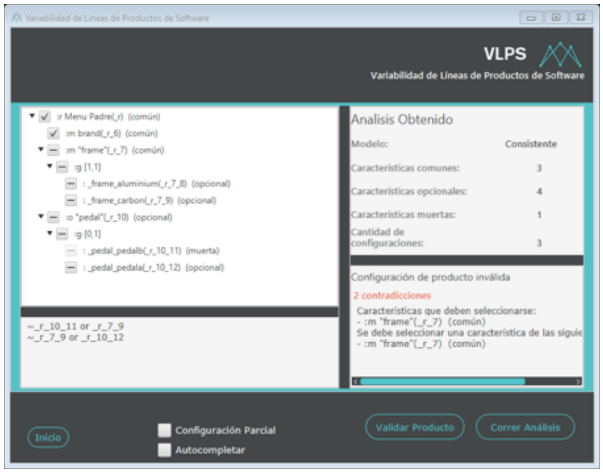

Figura 9. Validación de configuración.

Para evitar la pérdida de información, se debería haber realizado el análisis directamente sobre los modelos de variabilidad en Kconfig. Esto representa un tema de estudio aparte ya que supone un mayor esfuerzo al no contar con una especificación formal de la semántica de Kconfig y al no poder reusar herramientas, bibliotecas de código, existentes para el análisis de modelos de características.

5. Conclusiones

La industria de software ha comenzado a centrar el desarrollo de sus productos siguiendo un enfoque de líneas de productos de software. En este tipo de desarrollo la industria del software requiere de herramientas que permitan especificar la variabilidad de sus productos. En la actualidad, existe una marcada tendencia al empleo del lenguaje Kconfig para expresar tal variabilidad. Asimismo, el alto nivel de variabilidad y las interdependencias entre características posibles en los distintos productos requieren de herramientas que brinden

soporte en el análisis de la variabilidad, identificando posibles inconsistencias. En la literatura existen distintas propuestas para realizar tales análisis, una buena discusión se presenta en el trabajo de Benavides y colab. [19]. Sin embargo, estas alternativas están basadas en modelos de características, no incluyendo a modelos expresados en Kconfig [14], por eso se requiere de herramientas que permitan el análisis de la variabilidad. Actualmente, existen proyectos formulados (están en ejecución) para el desarrollo de herramientas que aborden esta problemática, por ejemplo, en https://kernelnewbies.org/KernelProjects/kconfig-sat se propone una herramienta basada en un SAT-solver. Incluso, en la última actualización de la definición de Kconfig se formula la necesidad de tales herramientas [13]. La herramienta desarrollada en este proyecto es un primer paso para satisfacer esta necesidad. La herramienta permite analizar modelos parciales especificados en Kconfig mediante su traducción a modelos de características.

Asimismo, la arquitectura de "pipe and filter" propuesta en este trabajo para el desarrollo de la herramienta permite emplear y/o explorar otro tipo de análisis a los modelos de variabilidad, facilitando así actividades futuras de investigación. Por ejemplo, es posible reemplazar el Analizador (Figura 4) por otro componente que realice los análisis sin emplear un SAT-solver. Una posibilidad sería emplear la herramienta FM2PN [24]. Incluso, se podría tomar la salida del Parser (Figura 4) para realizar un análisis directo sobre el modelo de variabilidad en Kconfig.

6. Agredecimientos

Este trabajo ha sido financiado en forma conjunta por CONICET y la Universidad Tecnológica Nacional. Se agradece el apoyo brindado por estas instituciones.

7. Referencias

[1] Clements, P., Northrop, L., *Software Product Lines – Practice and Patterns*, Addison-Wesley, 2001.

[2] Pohl, K., Böckle, G., van der Linden, F., *Software Product Line Engineering – Foundations, Principles, and Techniques*, Springer, 2005.

[3] Apel, S., Batory, D., Kästner, C., Saake, G., *Feature-Oriented Software Product Lines*, Springer-Verlag, 2013.

[4] Svahnberg, M., van Gurp, J., Bosch, J., "A Taxonomy of Variability Realization Techniques", *Software – Practice and Experience*, 35, 8, 2006, pp. 705-754.

[5] Gnesi, S., Jarzabek, S., "Special section on the 17th International Software Product Line Conference", *International Journal on Software Tools for Technology Transfer*, 17, 2015, pp. 555-557.

[6] von Rhein, A., Grebhahn, A., Apel, S., Siegmund, N., Beyer, D., Berger, T., "Presence-condition simplification in highly configurable systems", *International Conference on Software Engineering (ICSE)*, 2015, pp. 178-188.

[7] Metzger, A., Pohl, K., "Variability Management in Software Product Line Engineering", *International Conference on Software Engineering (ICSE)*, 2007.

[8] Berger, T., Pfeiffer, R., Tartler, R., Dienst, S., Czarnecki, K., Wasowski, A., She, S., "Variability mechanisms in software ecosystems", *Information and Software Technology*, 54, 2014, pp. 1520-1535.

[9] Berger, T. , Lettner, D., Rubin, J., Grünbacher, P., Silva, A., Becker, M., Chechik, M., Czarnecki, K., "What is a feature? A qualitative study of features in industrial software product lines", *Software product line conference (SPLC)*, 2015, pp. 16-25.

[10] Kang, K., Cohen, S., Hess, J., Nowak, W., Peterson, S., *Feature-oriented domain analysis (FODA) feasibility study*, Technical Report CMU/SEI-90-TR-21, CMU-SEI, 1990.

[11] Schmid, K., Rabiser, R., Grünbacher, P., "A comparison of decision modeling approaches in product lines", *Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, 2011, pp. 119-126.

[12] OMG, *Common Variability Language (CVL)*, OMG Revised Submission, disponible en: http://www.omgwiki.org/variability/lib/exe/fetch.php?media=cvl-revised-submission.pdf, 2012.

[13] Zippel, R., y otros autores, *kconfig-language.txt*, Disponible en https://github.com/torvalds/linux/blob/master/Documentation/kbuild/kconfig-language.txt, accedido 2017.

[14] Berger, T., She, S., Lotufo, R., Wasowski, A., Czarnecki, K., "A study of variability models and languages in the systems software domain", *IEEE Transaction on Software Engineering*, 39, 12, 2013, pp 1611-1640.

[15] Dintzner, N., van Deursen, A., Pinzger, M., "Analysing the Linux kernel feature model changes using FMDiff", *Software & Systems Modeling*, doi:10.1007/s10270-015-0472-2, 2015.

[16] Berger, T., Rublack, R., Nair, D., Atlee, J. M., Becker, M., Czarnecki, K., Wasowski, A., "A survey of variability modeling in industrial practice", *Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, 2013.

[17] Passos, L., Teixeira, L., Dintzner, N., Apel, S., Wąsowski, A., Czarnecki, K., Borba, P., Guo, J., "Coevolution of variability models and related software artifacts. A fresh look at evolution patterns in the Linux kernel", *Empirical Software Engineering*, 21, 4, 2016, pp. 1744-1793.

[18] Abal, I., Brabrand, C., Wasowski, A., "42 Variability Bugs in the Linux Kernel: A Qualitative Analysis", *ASE'14*, 2014, pp. 421-432.

[19] Benavides, D., Segura, S., Ruiz-Cortés, A., "Automated analysis of feature models 20 years later: A literature review", *Journal of Information Systems*, 35, 2010, pp. 615-636.

[20] Mendonca, M., Branco, M., Cowan, D., "S.P.L.O.T.: Software Product Lines Online Tools", *Proc. 24th ACM SIGPLAN Conf. OOPSLA*, 2009.

[21] She, S., Lotufo, R., Berger, T., Wasowski, A., Czarnecki, K., "The Variability Model of The Linux Kernel". *Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, 2010.

[22] Sincero, J., Schirmeier, H., Schröder-Preikschat, W., Spinczyk, O., "Is The Linux Kernel a Software Product Line?", *SPLC-OSSPL 2007*, 2007.

[23] Parnas, D.L., "On the Design and Development of Program Families", *IEEE Transactions on Software Engineering*, 2, 1, March 1976, pp: 1-9.

[24] Duttweiler, J., *F2MPN (Feature Models 2 Petri Net)*, Proyecto final de Carrera, Universidad Tecnológica Nacional, Facultad Regional Santa Fe, 2016.