Análisis de la Variabilidad de Líneas de Productos de Software Especificadas en Kconfig.

Resumen: *Kconfig fue creado para describir la variabilidad del kernel de Linux y, desde entonces, diversos proyectos de desarrollo de código abierto han adoptado este lenguaje. Estos proyectos son altamente configurables y están conformados por un gran número de características o propiedades. Las características en un sistema configurable interactúan de una manera no trivial, y esto puede introducir errores en los productos derivados. Por otro lado, existen herramientas, tal como SPLOT, que soportan el análisis de modelos de features, permitiendo la detección de inconsistencias, detección de características muertas (las cuales no pueden incluirse en ningún modelo) y el número de productos derivables, entre otras funciones. A partir del escenario planteado, en este trabajo se propone la traducción de un modelo de variabilidad especificado en Kconfig a un modelo de características especificado en SPLOT, para poder luego obtener información sobre las propiedades de interés mencionadas anteriormente.*

Palabras Claves: *Kconfig; variabilidad; característica; modelo de características.*

Abstract: *Kconfig was created for describing the Linux kernel variability and, since then, various open source code development projects have adopted this language. These projects are highly configurable and are shaped by a big number of features or properties. Features in a configurable system interact in a non-trivial way, and this may introduce errors in products. On the other hand, there are tools like SPLOT that supports feature models analysis, allowing inconsistencies detection, dead features detection (which can't be included in any model) and the number of derived products, among other functions. Since the exposed scenario, this paper proposes the translation of a variability model specified in Kconfig to a feature model specified in SPLOT in order to obtain information from properties of interest mentioned previously.*

Keywords: *Kconfig; variability; feature; feature model.*

Matias Sequeira, Rocío González

Instituto INGAR–UTN, Conicet - Avellaneda 3657, Santa Fe.

Facultad Regional Santa Fe, UTN

E-mail de contacto: matias-sequeira@hotmail.com, rociogonzalez@outlook.es

INTRODUCCIÓN

Una tendencia creciente en el desarrollo de software es la necesidad de desarrollar múltiples productos de software similares en vez de un producto individual. Hay varias razones para esto: los productos pueden estar enfocados a distintos sectores del mercado, estar sujetos a distintas restricciones legales o culturales, o deben satisfacer necesidades específicas de diferentes stakeholders. Debido a las restricciones de costo y tiempo, no es posible desarrollar un nuevo producto desde cero para cada cliente, y el re-uso de software debe ser incrementado.

Frente a este desafío, la ingeniería de líneas de productos de software (SPLE), surge como un paradigma viable e importante, que permite a las empresas desarrollar familias de productos, disminuyendo costos y tiempos, basándose en el re-uso de componentes (Clements y Northrop ,2001); (Pohl et al., 2005).

Una línea de productos de software (SPL) es una familia de sistemas de software desarrollados a partir de un conjunto de características comunes, que apunta a satisfacer necesidades específicas de un segmento de mercado. Una SPL 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 la aplicación (Clements y Northrop, 2001); (Pohl et al., 2005).

Muchos proyectos de desarrollo de software deben administrar una variabilidad muy grande. Proyectos que adoptan SPL emplean el concepto de variabilidad para derivar productos de software individuales en nicho de mercados (Apel et al., 2013). Los modelos de variabilidad representan las características, o "features", comunes y variables de productos en una SPL (Kang et al., 1990); (Kang et al., 2002). Por otro lado, existen proyectos de software altamente configurables, como es el caso del kernel de Linux, donde las opciones de configuración, referenciadas en este trabajo como características, son empleados para derivar el producto cumpliendo ciertas propiedades funcionales y no-funcionales, según las necesidades del usuario.

Sistemas altamente configurables pueden llegar a tener un gran número de características. Reportes de sistemas industriales indican poseer ciento de características (Berger et al., 2013a) y en sistemas de código abierto, la cantidad de características asciende a miles (Berger et al., 2013b).

Las características en un sistema configurable interactúan de una manera no trivial, y su interacción puede introducir errores en los productos derivados (Abal et al., 2014). El número de configuraciones es exponencial en el número de características, por lo que no es posible analizar cada configuración en forma separada. En los últimos años se identificaron varias funciones que son de utilidad para extraer información desde los modelos de características (Feature Model), tales como detección de inconsistencias, detección de características muertas (las cuales no pueden incluirse en ningún modelo), número de productos derivables, entre otras (Kang et al., 1990); (Benavides et al., 2010). Debido al incremento en tamaño y complejidad de los modelos, surge el desafío de proporcionar un soporte automático para llevar adelante estas funciones. Los enfoques actuales usan lógica y problemas de satisfacción de restricciones (CSP). Las distintas propuestas están optimizadas para modelos con ciertas propiedades, tales como tamaño y densidad de restricciones, debido a la complejidad computacional de los problemas de configuración. Muchas de estas propuestas están disponibles en el ambiente SPLOT (www.splot-research.org) (Mendonca et al., 2009). Sin embargo, los ejemplos disponibles son casos de estudios pequeños y no reflejan modelos del mundo real.

Kconfig (Zippel et al., 2015), fue creado para

describir la variabilidad del kernel de Linux y diversos proyectos de desarrollo de código abierto han adoptado este lenguaje para definir su variabilidad (Berger et al., 2013); (She et al., 2010). El modelo de variabilidad especificado en Kconfig puede ser interpretado como un modelo de características (She et al., 2010); (Sincero et al., 2007). A partir de lo enunciado, en este trabajo se presenta el desarrollo de una herramienta que permite traducir un modelo de variabilidad expresado en Kconfig en un modelo de características. El modelo traducido es representado según la notación empleada en SPLOT. Esto permite luego el análisis de ciertas propiedades del modelo de variabilidad.

A continuación se introducen los conceptos empleados en el trabajo y la representación de Kconfig mediante Feature Model. Luego, se describen los resultados obtenidos, la discusión del tema en cuestión. Por último, se presentan las conclusiones del trabajo.

DESARROLLO

En este trabajo se propone la traducción de un modelo de variabilidad especificado en Kconfig a un modelo de características especificado en SPLOT, para poder luego analizar su variabilidad. En consecuencia, en primer lugar se analizan ambos modelos de representación: el modelo de características y su especificación en SPLOT, y el modelo de variabilidad en Kconfig. Luego, se incluye la representación de Kconfig mediante un modelo de características.

Modelo de Características

Un modelo de características es una representación compacta de todos los productos de una SPL 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 (Kang et al., 1990).

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 un característica hijo con su padre, indicando que el hijo aparecerá en todos los productos en los que el padre esté incluido.
- Opcional: Relación que vincula una característica hijo con su padre, indicando que el hijo podrá opcionalmente estar incluidos en algunos productos en los que esté incluido su padre.
- Cardinalidad grupal: La cardinalidad grupal es un intervalo denotado por [min ... máx] que limita el número de características hijo que pueden ser incluidas en un producto, cuando su padre está incluido, siendo los límites inferior y superior respectivamente.
- Requiere: Esta relación indica una implicación entre una característica requerida y una característica restringida. 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 1 se ilustra un ejemplo de un modelo de característica, el cual es ilustrado empleando la herramienta SPLOT. 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 1 indican una relación de obligatoriedad entre las características padre e hijas. Los círculos

blancos de la Figura 1 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 (Figura 1).

Según el modelo de la Figura 1, toda arquitectura Rapid IO 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.

En SPLOT las relaciones requiere y excluye se las representan por fórmulas proposicionales en forma normal conjuntiva. Si una característica A requiere una característica B, se expresa ⌐A ∨ B. En cambio, si una característica A excluye una característica B, se especifica ⌐ A ∨ ⌐ B.

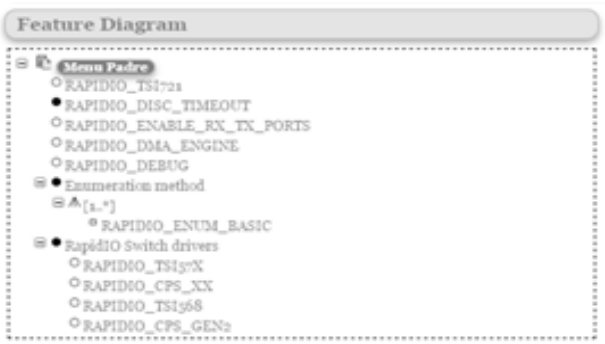

Figura 1. Ejemplo de diagrama de características.

Lenguaje Kconfig

En Linux se denomina configs a las opciones de configuración (características). Estas opciones y las dependencias entre las mismas se definen mediante el lenguaje Kconfig. Configs pueden estar anidadas dentro de otras configs (menuconfig); pueden estar agrupadas en menús (menu), en grupos de opciones (choice) o bloques if. Estos agrupadores pueden anidarse entre sí, de manera indistinta. El configurador del kernel, xconfig, presenta el modelo kconfig como un árbol de opciones, donde el usuario selecciona las configuraciones que desea para construir el kernel. En caso de existir relaciones transversales en el árbol, se visualizan en la parte inferior de la interfaz de xconfig.

La Figura 2 muestra un fragmento de un modelo de variabilidad de Linux expresado en el lenguaje Kconfig. El fragmento contiene tres configs de tipo Boolean, un menuconfig seguido de la sentencia if, el bloque if contiene un menu compuesto por tres configs de tipo Tristate.

```
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 2. Fragmento de un modelo de variabilidad en Kconfig.

Los menús son empleados para agrupar configs. Kconfig provee un mecanismo de visibilidad condicional para menús. Si la condición es falsa, el menú y sus hijos son descartados por el configurador. En la Figura 2 se incluye un menú, 1-wire Bus Masters, que agrupa tres configs.

La Figuras 3 incluye otro fragmento que ilustra otros constructores de Kconfig, se incluye un menuconfig y un bloque if que contiene una config seleccionable

(prompt) de tipo Boolean y un grupo de opciones (choice) la cual posee 2 configs del tipo Boolean.

```
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 3. Fragmento de un modelo de variabilidad en Kconfig, ejemplo de menuconfig y choice, Kernel compression mode.

Cada config posee un nombre y un tipo. Los tipos pueden ser: boolean, tristate, integer (int o hex), o string (Zippel et al., 2015; She et al., 2010). Una config 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 de la parte 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.

Integer configs se emplean para especificar opciones numéricas, tal como el tamaño de un buffer. String configs permiten especificar el nombre de un elemento configurable, como puede ser el nombre de una partición del disco.

She et al. (2010) denominan a las configs de tipo boolean y tristate como "switch configs", en cambio a las config de tipo integer y string la denominan "entry-field config".

Una config puede ser seleccionable por el usuario o no. La opción es seleccionable si el tipo de la config es seguido por un prompt, una breve explicación de la opción, como la config THERMAL_HWMON en la Figura 3.

Una especificación depends-on introduce una dependencia que debe ser satisfecha cuando se selecciona la config. Por ejemplo, en la Figura 2 W1_MASTER_MXC depende de las opciones MMU y SBUS (ambas opciones deben ser seleccionadas). Inversamente, una especificación select obliga la selección de otra config cuando la config es seleccionada por el usuario. Por ejemplo, en la Figura 2, cuando se selecciona W1_MASTER_DS2482, se debe seleccionar X86_HT.

Depends-on es también empleada para especificar anidamiento entre opciones, es decir, aquellas configs que sólo dependen de un solo objeto se podrían pensar como hijas de ese objeto.

Los grupos de opciones (choice configs) permiten definir alternativas. Las opciones pueden ser boolean or tristate. Cuando se indica que la opción es seleccionada (y), se debe seleccionar una única opción (XOR); cuando el valor es m, se puede seleccionar uno o más opciones (OR). La choice de la Figura 3 es un ejemplo de XOR. Un grupo de opción marcado como opcional puede ser configurada con el valor n. En ese caso no es necesario seleccionar algunas de sus opciones. Sin embargo, un grupo de opciones sin una marca es considerado obligatorio y no se le puede asignar el valor n.

Un bloque if permite configurar los elementos que contiene dentro cuando la expresión, es decir, la config evalúa verdadera. Por esto, se podría pensar que la especificación if agrega una dependencia entre los componentes del bloque y la config de la expresión que evalúa. En la Figura 2 W1_CON y las config agrupadas en el menú "1-wire Bus Masters" dependen del valor que posea la config W1 y en la Figura 3 se puede ver otro ejemplo donde THERMAL_HWMON y la choice están condicionadas por el valor de THERMAL.

Representación de Kconfig mediante Feature Model

La herramienta propuesta en este trabajo permite automatizar el proceso de construcción de un modelo de características (Feature model) a partir de un modelo de variabilidad expresado en Kconfig. El modelo de característica resultante es especificado empleando la notación de la herramienta SPLOT, el cual es un archivo de tipo SXFM (almacenado en formato xml).

La Figura 4 ilustra la arquitectura de tipo Pipe & Filter propuesta para la herramienta. La entrada a la herramienta es un archivo especificado en Kconfig (archivo de texto). Las Figuras 2 y 3 son fragmento de ejemplo de la entrada a la herramienta. El primer componente es un Parser (Figura 4) que genera las instancias del modelo conceptual de Kconfig (Figura 5).

Luego, el Transformador (Figura 4) aplica un conjunto de reglas para traducir las instancias del modelo de Kconfig (Figura 5) en instancias del modelo de características. Por último, el Generador crea el archivo de salida especificado en SXFM (archivo XML).

Reglas para traducción de lenguaje KConfig a Splot

Para poder obtener un modelo de características desde un modelo de variabilidad expresado en Kconfig, analizamos el lenguaje Kconfig, definiendo su semántica. La Figura 5 representa los conceptos de Kconfig considerados en este trabajo.

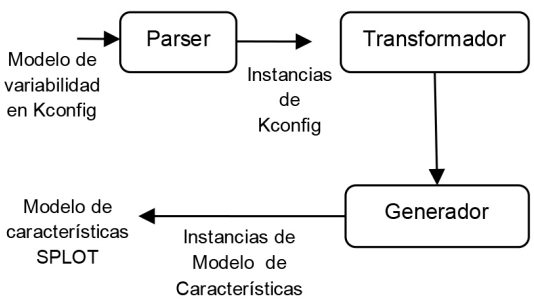

Figura 4. Arquitectura de la herramienta propuesta.

Luego, definimos reglas que permiten mapear conceptos de Kconfig a conceptos del modelo de característica.

- Como raíz (_r), se crea un menú (Menu en Figura 5) ficticio que contiene a los elementos raíz del archivo KConfig.

- Aquellas configs (Config en Figura 5) que son del tipo "bool" o "tristate" y poseen un "prompt" son traducidas como configs opcionales (:o). El resto son tomadas como obligatorias (:m).

- Se considera que todo lo que está dentro de un menú, pasa a ser elemento hijo del mismo (configs, menuconfigs, if, choices, y otros menús).

- Las choices (Choice en Figura 5) son traducidas según el "type" de la misma: si son del tipo "bool", sólo puede seleccionarse un componente dentro de la misma [1..1]; en caso de ser del tipo "tristate", se pueden seleccionar uno o más componentes con el valor "m" [1..*]. Si la choice posee "OPTIONAL" dentro de sus opciones, significa que las restricciones anteriores permiten que no se seleccionen componentes (que tomen valor n), quedando [0..1] y [0..*], respectivamente.

- Los menuconfigs (Menuconfig en Figura 5) son traducidos de forma similar a las configs, ya que son configs que agrupan otras configs.

- Cuando una entrada posee la opción "depends on" (DependsOn en Figura 5) y la expresión que le sigue hace referencia a sólo una entrada (un menú, una config, etc.), se traduce como hija de la entrada referenciada en la expresión.

Por lo contrario, si la expresión hace referencia a más de una entrada, se considera como una equivalencia con la entrada referenciada. La herramienta lo traduce en fórmulas proposicionales en forma normal conjuntiva para cada dirección de la implicancia. En la Figura 6 se muestran las equivalencias lógicas que

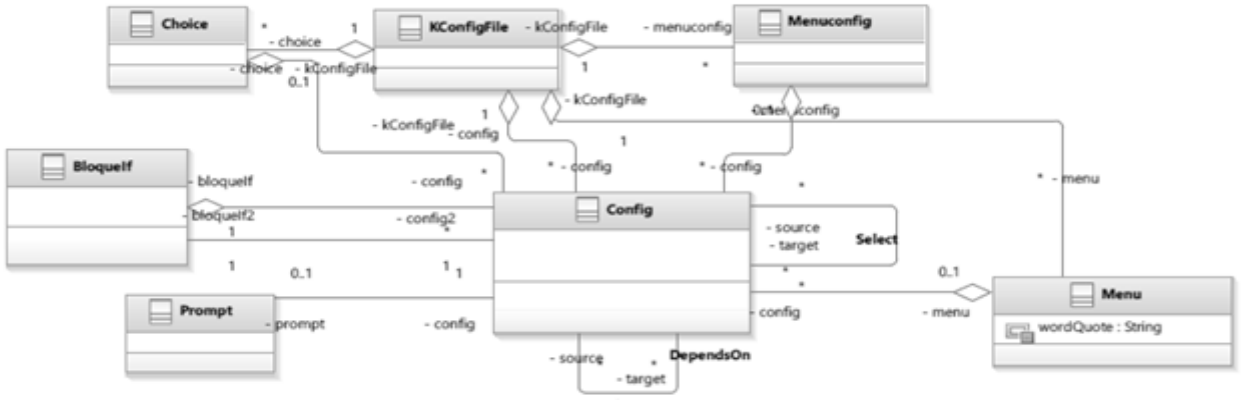

Figura 5. Modelo Conceptual de Kconfig.

se trabajan, en la parte superior se ilustra la transformación desde la definición del depends on hacia la equivalencia (A en Figura 6), en la parte inferior se incluye la transformación desde la equivalencia en un conjunto de fórmulas proposicionales en forma normal conjuntiva (B en Figura 6).

- Cuando una config posee la opción "select" (Select en Figura 5) precedida por una expresión, se traduce como una restricción donde la config implica la expresión. Si al final de esta opción se encuentra un "if", la entrada que continua al if implica la restricción declarada.

- Los elementos de un bloque if (BloqueIf en Figura 5) (configs, menuconfigs, menu, choice y otros bloques if), son considerados como hijos de la config o menuconfig mencionada en la entrada "if".

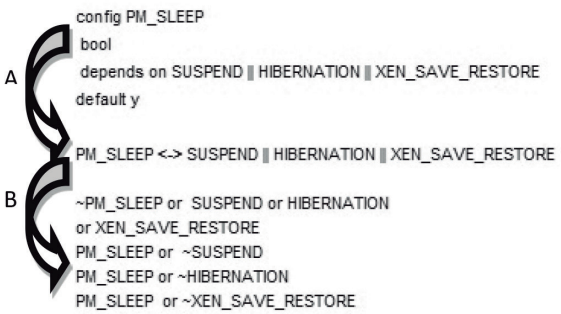

Figura 6. Ejemplo de equivalencia lógica en la expresión depends on.

RESULTADOS

La herramienta propuesta permite realizar la traducción automática de modelos de variabilidad expresados en Kconfig en modelos de características. Sobre los modelos resultantes es posible aplicar las herramientas disponibles en la actualidad para analizar la variabilidad de la SPL. Cabe destacar que sólo un subconjunto de modelos de Kconfig es traducible a modelos de características, debido a que existen constructores que no tienen semántica definida en un modelo de característica.

Para la implementación del Parser (Figura 4) se empleó el generador de parser ANTLR 4 (Parr, 2014). En la implementación se realizó un pre-procesamiento a los modelos en Kconfig. Esto se debe a que un modelo de variabilidad expresado en Kconfig puede estar descompuesto en múltiples archivos. Los archivos son enlazados mediante la primitiva source. De esta manera en el pre-procesamiento se navegó por los distintos archivos obteniendo un único archivo con el modelo completo de variabilidad. Además, en Kconfig es posible incorporar anotaciones que ayuden a interpretar su especificación, estas anotaciones son incorporadas mediante el constructor help, ---help---, o el

símbolo #. En el pre-procesamiento las anotaciones de help son eliminadas.

Utilizando el modelo de Kconfig mostrado en la Figura 2 como entrada a la herramienta, obtenemos el modelo de característica en formato SXFM (Figura 7). En la Figura 8 se ilustra el modelo utilizando la herramienta SPLOT, donde se destacan las características que son inherentes a todas las configuraciones posibles.

En la Figura 9 se observa que el modelo posee 10 características, de las cuales 5 son opcionales, y 4 son obligatorias (1 es la raíz). En el modelo se definieron 4 restricciones, las cuales representan relaciones del tipo requiere o excluye. El modelo no posee características muertas (dead features) y el número de características comunes es 7. Se puede ver en la Figura 8 que características opcionales, como W1, y W1_MASTER_MXC, no lo son en la realidad, ya que siempre van a ser características incluidas en todo producto derivado.

En las Figuras 10 se puede ver la conversión que se obtiene del fragmento de la Figura 3. Esta figura muestra la representación en SPLOT y el resultado de su análisis.

```
:r Menu Padre(_r)
        :m X86_HT(_r_6)
        :m MMU(_r_7)
        :m SBUS(_r_8)
        :o W1(_r_9)
                :o W1_CON(_r_9_10)
                :m "1-wire Bus Masters"(_r_9_11)
                        :o W1_MASTER_DS2490(_r_9_11_12)
                        :o W1_MASTER_DS2482(_r_9_11_13)
                        :o W1_MASTER_MXC(_r_9_11_14)

~_r_9_11_13 or _r_6
~_r_9_11_14 or _r_7 or _r_8
_r_9_11_14 or ~_r_7
_r_9_11_14 or ~_r_8
```

Figura 7. Modelo de característica para la variabilidad definida en Kconfig en la Figura 2. Representación en formato SXFM.

Feature Tree:

- Menu Padre (Common)
  - X86_HT (Common)
  - MMU (Common)
  - SBUS (Common)
  - W1 (Common)
    - W1_CON
    - "1-wire Bus Masters" (Common)
      - W1_MASTER_DS2490
      - W1_MASTER_DS2482
      - W1_MASTER_MXC (Common)

Extra Constraints:

-constraint_1: (~_r_9_11_13) or _r_6
-constraint_2: (~_r_9_11_14) or _r_7 or _r_8
-constraint_3: _r_9_11_14 or (~_r_7)
-constraint_4: _r_9_11_14 or (~_r_8)

Figura 8. Modelo de característica para la variabilidad definida en Kconfig en la Figura 2. Representación gráfica en SPLOT.

Statistics	
#Features	10
- Optional	5
- Mandatory	4
- Grouped	0
- Groups	0
Tree Depth	3
ECR (%)	50
#Extra constraints	4
#Distinct extra constraints variables	5
Clause Density	0,8
#CNF Clauses	18

Debugging Analyses (Click to Run the SAT solver)	
Consistency	consistent
Running Time (ms)	7
#Dead Features	0
- Running Time (ms)	4
#Common Features	7 view
- Running Time (ms)	6

Figura 9. Análisis de la variabilidad del ejemplo de Figura 2.

DISCUSIÓN

En la sección previa se incluyeron ejemplos sencillos que ilustran el proceso de traducción. Sin embargo, se está trabajando sobre la versión 4.2 de Linux para la arquitectura x86. El archivo raíz de Kconfig está disponible en https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/plain/arch/x86/Kconfig. El modelo posee más de 10000 cláusulas config, 200 menuconfig, 170 menu, 60 choice, 6900 definiciones de select, y 9000 depends on. Pruebas similares se reportan en la literatura con la versión 2.6.28 de la arquitectura x86 de Linux, identificando 5300 config, 71 menu, 32 choice (She et al., 2010). Esto fue traducido a 5426 características. La propuesta de She et al. (2010) identificó la necesidad del estudio de la variabilidad en Linux, siendo este un muy buen ejemplo de un modelo de variabilidad a gran escala empleado en la práctica. Además, en ese trabajo se muestra la diferencia existente entre los modelos académicos reportados en bases como la de SPLOT (Mendonca et al., 2009) y modelos reales, requiriendo el desarrollo de nuevas herramientas para el análisis de variabilidad. El mayor problema en el análisis de la variabilidad con herramientas que usan lógica y problemas de satisfacción de restricciones (CSP), como SPLOT, radica en la imposibilidad de analizar problemas con un gran número de características y de relaciones requiere y excluye. Como alternativa a este tipo de herramientas Martinez et al. (2014), Martinez et al. (2013) y Duttweiler (2014) propusieron emplear el formalismo de redes de Petri para representar la variabilidad. A partir de las propiedades de las redes de Petri definidas, tales como alcanzabilidad, acotación y distancia sincrónica, entre otras, es posible abordar problemas de inviabilidad en una SPL y obtener información cuantitativa de los modelos de variabilidad. En consecuencia, a partir de estos trabajos, se explorará la generación automática de las redes de Petri a partir de los modelos de variabilidad en Kconfig con el objeto de evaluar la factibilidad de analizar un gran número de característica y poder comparar los resultados con los análisis realizados con SPLOT.

Statistics	
#Features	6
- Optional	2
- Mandatory	1
- Grouped	2
- Groups	1
Tree Depth	4
ECR (%)	33
#Extra constraints	1
#Distinct extra constraints variables	2
Clause Density	0.5
#CNF Clauses	10
Debugging Analyses (Click to Run the SAT solver)	
Consistency	consistent
Running Time (ms)	58
#Dead Features	0
- Running Time (ms)	16
#Common Features	1 view
- Running Time (ms)	0

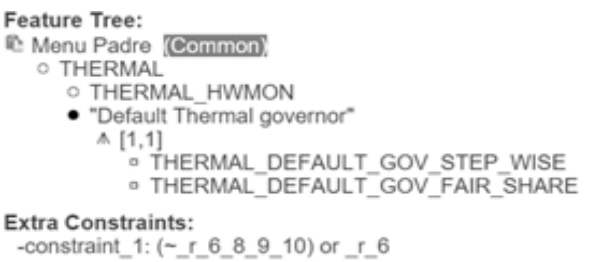

Figura 10. Modelo de característica y análisis de la variabilidad para la Figura 3. Representación gráfica en SPLOT.

CONCLUSIÓN

La herramienta propuesta permite analizar propiedades de modelos de variabilidad especificados en Kconfig. Este análisis es posible a partir de la traducción de los modelos en Kconfig al formato requerido por herramientas de análisis de variabilidad existentes. Sin embargo, dado el tamaño de los modelos en Kconfig estas herramientas pueden no ser adecuadas para analizar el modelo de variabilidad completo. Por tal motivo se está trabajando en: i) la integración de la propuesta a otras herramientas de análisis, tales como las especificadas en la sección 4, discusión (Martinez et al., 2014; Martinez et al., 2013; y Duttweiler, 2014); ii) la implementación de un modelo de análisis de la variabilidad especificada en Kconfig sin requerir la traducción a un modelo de características. La última de estas líneas de trabajo permitiría no perder información no soportada por los modelos de características.

RECONOCIMIENTOS

Se agradece el apoyo financiero brindado por la Universidad Tecnológica Nacional (PID25/O144).

REFERENCIAS

Abal, I., Brabrand, C., Wasowski, A. (2014). 42 Variability Bugs in the Linux Kernel: A Qualitative Analysis. ASE'14, 421-432.

Apel, S., Batory, C., Kästner, C., Saake, G. (2013). Feature-Oriented Software Product Lines, Springer-Verlag.

Benavides, D., Segura, S., Ruiz-Cortés, A. (2010). Automated analysis of feature models 20 years later: A literature review. Journal of Information Systems, 35, 615-636.

Berger, T., Rublack, R., Nair, D., Atlee, J.M., Becker, M., Czarnecki, K., Wasowski, A. (2013a). A survey of variability modeling in industrial practice. VaMoS. ACM.

Berger, T., She, S., Lotufo, R., Wasowski, A., Czarnecki, K. (2013b). A study of variability models and languages in the systems software domain. IEEE Transaction on Software Engineering, 39 (12), 1611-1640.

Clements, P., Northrop, L. (2001).Software Product Lines – Practice and Patterns. Addison-Wesley.

Duttweiler, J. (2014). Desarrollo de una herramienta para analizar la variabilidad en Líneas de Productos de Software. 2do Congreso Nacional de Ingeniería Informática / Sistemas de Información (CoNaIISI 2014).

Kang, K., Cohen, S., Hess, J., Nowak, W., Peterson, S. (1990). Feature-oriented domain analysis (FODA) feasibility study, Tech.Rep.CMU/SEI-90-TR-21, CMU-SEI.

Kang, K., Lee, J., Donohoe, P. (2002). Featured-Oriented Product Line Engineering. IEEE Software, 19(4), 58-65.

Martinez, C., Díaz, N., Gonnet, S., Leone, H. (2014). A Petri Net Variability Model for Software Product Lines. Electronic Journal of SADIO, 13 (1).

Martinez, O.C., Gonnet, S., Leone, H. (2013). A Petri Net approach for representing Orthogonal Variability Models. International Journal of Computers & Technology, Council for Innovative Research, 9 (1), 995- 1003.

Mendonca, M., Branco, M., Cowan, D. (2009). S.P.L.O.T.: Software Product Lines Online Tools. Proc. 24th ACM SIGPLAN Conf. Companion Object Oriented Programming Systems Languages and Applications (OOPSLA). ACM.

Parr, T. (2013). The Definitive ANTLR 4 Reference, Pragmatic Bookshelf, 2nd edition.

Pohl, K., Böckle, G., Van der Linden, F. (2005). Software Product Line Engineering – Foundations, Principles, and Techniques. Springer.

She, S., Lotufo, R., Berger, T., Wasowski, A., Czarnecki, K. (2010). The Variability Model of The Linux Kernel. VaMoS. ACM.

Sincero, J., Schirmeier, H., Schröder-Preikschat, W., Spinczyk, O. (2007). Is The Linux Kernel a Software Product Line?. Workshop SPLC-OSSPL 2007.

FECHA DE PRESENTACIÓN: 22/02/16
FECHA DE APROBACIÓN: 10/08/16