Modelando la Variabilidad de Características Complejas en Líneas de Productos Software

José Miguel Horcas, Mónica Pinto, and Lidia Fuentes

Universidad de Málaga, CAOSD Group, Málaga, Spain
{horcas,pinto,lff}@lcc.uma.es http://caosd.lcc.uma.es/

Resumen A pesar de la gran cantidad de trabajos y de herramientas existentes en el ámbito de las Líneas de Productos Software, muchos dominios de aplicación actuales como la ingeniería web no pueden verse beneficiados del uso de una Línea de Producto Software. Además, fuera del mundo académico, la empresa sigue siendo bastante reticente a usar este enfoque. En este artículo reflexionamos sobre las razones de esta falta de adopción, tanto desde el punto de vista de las características complejas que son imprescindibles en los sistemas actuales, como del soporte que las herramientas existentes proporcionan.

Keywords: Líneas de Productos Software · Modelado · Soporte de herramientas · Variabilidad

1. Introducción

Las Líneas de Productos Software (SPL, de *Software Product Line*) [38] se han vuelto muy complejas debido a los exigentes requisitos de los actuales dominios de aplicación. Por ejemplo, si pensamos en aplicaciones para el Internet de las Cosas (IoT, de *Internet of Things*) o en sistemas ciberfísicos (CPS, de *CiberPhysical System*), aún sin tener en cuenta los requisitos específicos de la aplicación a desarrollar, aparecen numerosos requisitos del dominio de aplicación para los que las propuestas actuales de SPL, y en especial las herramientas que las soportan, no están preparadas [5,40].

Por un lado, al margen de la variabilidad propia de las aplicaciones, la variabilidad en muchos dominios actuales aparece en muchas otras dimensiones [30], como es en los dispositivos hardware sobre los que se desplegará la aplicación y en cómo pueden configurarse, en los despliegues alternativos que pueden llevarse a cabo tanto desde el punto de vista hardware como software y en como afectan a los atributos de calidad del sistema, o en la variabilidad del contexto de ejecución. Es necesario por tanto reflexionar sobre todas estas dimensiones de variabilidad, sobre si los modelos de variabilidad existentes son los apropiados para todas las dimensiones que es necesario modelar, sobre cuáles son las relaciones entre ellas y como se modelan [39]. Si pensamos por ejemplo en los modelos de características actuales y en las herramientas existentes, sin soporte para características clonables [19] no es posible modelar la variabilidad de un sistema IoT o CPS con múltiples dispositivos iguales (e.g., distintos sensores) [4], donde el número de sensores a desplegar variará para cada sistema y donde los parámetros de cada sensor se configurarán de forma diferente. Tampoco es posible definir y razonar sobre muchos de los parámetros de configuración de los dispositivos hardware de un CPS sin características numéricas y sin razonadores que sean capaces de manejar la complejidad añadida por estas características numéricas a la hora de resolver el modelo de variabilidad [26].

Por otro lado, la configuración de estos sistemas, tanto en tiempo de diseño como en tiempo de ejecución, debería estar dirigida mayormente por los atributos de calidad

que el sistema debe satisfacer (e.g., rendimiento, latencia, consumo de energía) [51]. Esto exige que las configuraciones de estos sistemas deben poder generarse optimizando uno o varios de estos atributos de calidad [25]. Para conseguirlo debe ser posible razonar sobre el impacto de las distintas características del modelo de variabilidad en los atributos de calidad que se estén considerando, debe ser posible identificar un deterioro en el cumplimiento de esos atributos de calidad durante la ejecución del sistema y debe ser posible adaptar el sistema al nuevo contexto automáticamente de forma que sus atributos de calidad sigan estando en un rango aceptable.

Aunque las características mencionadas anteriormente no son novedosas y hay multitud de propuestas de SPL que las tienen en cuenta [40], la realidad es que no están integradas en una sola propuesta que de respuesta a las necesidades de estos nuevos sistemas y, sobre todo, no están soportadas por las herramientas existentes. Cabe resaltar que a pesar de que existen más de un centenar de herramientas sobre SPL [5,34,37], hay aspectos sobre el modelado y resolución de la variabilidad que ninguna de ellas es capaz de cubrir correctamente. Esto hace que las empresas sean reticentes a adoptar un enfoque de SPL, al no contar con un soporte maduro de herramientas [40]. Por todos estos motivos, las SPLs no se utilizan fuera de su comunidad tanto como podría esperarse dado los beneficios que su uso proporcionaría en numerosos dominios de aplicación.

El objetivo de este trabajo es reflexionar sobre la complejidad de modelar ciertas características de variabilidad que actualmente exigen las aplicaciones y que las herramientas de SPL no soportan. Para ello se identifican las características más complejas del modelado de la variabilidad que existen hoy en día en la industria y se analiza el soporte actual de las herramientas existentes de referencia en SPL exponiendo las limitaciones más importantes de ellas. Este trabajo expone las dificultades de la industria para adoptar los enfoques SPL, y sienta las bases sobre las características que debería tener una herramienta para dar soporte completo a un proceso de SPL en la industria. A partir de este trabajo, la comunidad de SPL puede comenzar el desarrollo de los artefactos adecuados (e.g., metamodelos, lenguajes, formatos) para definir una herramienta que cubra todas las necesidades exigidas por los proyectos actuales de SPL en la industria.

El resto del artículo se estructura de la siguiente forma. La Sección 2 presenta el trabajo relacionado sobre modelado de la variabilidad y soporte de herramientas. La Sección 3 identifica las características de variabilidad en proyectos complejos y analiza el soporte de las herramientas actuales, discutiendo sus limitaciones y como debería ser el soporte ideal. Por último, la Sección 4 presenta las conclusiones y el trabajo futuro.

2. Trabajo Relacionado

Existen una gran cantidad de trabajos que estudian el modelado de la variabilidad [8,28,39,40]. Sin embargo, el soporte de herramientas para llevarla a cabo ha sido estudiado principalmente en revisiones sistemáticas [5,34,37].

2.1. Modelado de la variabilidad

La mayoría de los trabajos sobre modelado de la variabilidad exploran diferentes dimensiones y relaciones entre los elementos de modelado [28]. La Figura 1 muestra un resumen de los trabajos más importantes sobre modelado de variabilidad y modelos de características. Hay que resaltar que no están todos los trabajos existentes, sino que se han incluido principalmente aquellos donde aparecieron por primera vez las características discutidas a lo largo del artículo.

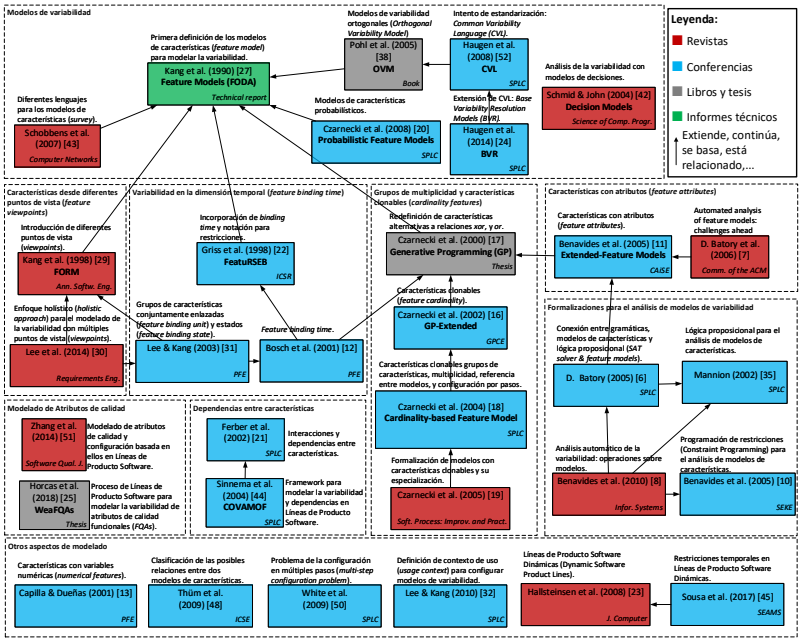

Figura 1. Resumen de los trabajos más importantes sobre modelado de variabilidad.

Desde la introducción de los modelos de características por Kang et al. en 1990 [27], infinidad de extensiones y lenguajes han sido propuestos [17,43]. Por un lado, no hay un estándar para el modelado de la variabilidad, por lo que han surgido distintas propuestas para modelar conceptos similares, como los modelos de variabilidad ortogonales [38], los modelos de características probabilísticos [20], o lo modelos de decisiones [42]. Aunque hubo un intento de estandarización con la definición de CVL [52] y BVR [24], esta no finalizó satisfactoriamente. Por otro lado, se han definido múltiples versiones de modelos de características "extendidos" [16], por ejemplo, para añadir atributos a las características [7,11], para definir grupos de multiplicidad y características clonables [18,19], para definir características numéricas [13], para modelar las interacciones y dependencias entre características [21,44,48], para considerar múltiples puntos de vista [29,30], para modelar atributos de calidad [25,51] o el contexto de uso [32]. Desafortunadamente, no existe una propuesta conjunta en la que poder utilizar todas estas características. Además, el problema se complica si tenemos en cuenta la dimensión temporal [12,22,31] o la configuración del modelo en múltiples pasos [50], o cuando queremos llevar esas características a tiempo de ejecución con Líneas de Productos Software Dinámicas (DSPL, de *Dynamic Software Product Lines*) [23], pudiendo incluso añadir restricciones temporales [45].

2.2. Soporte de herramientas para modelar la variabilidad

Existen varias revisiones sistemáticas que estudian el soporte de herramientas para SPLs [5,34,37]. El problema de estos trabajos es que estudian las herramientas desde el punto de vista de su documentación y/o artículo de referencia. Sin embargo, la mayoría de los detalles técnicos de las herramientas suelen estar en tesis, informes técnicos y páginas webs, que no suelen ser considerados estudios primarios en las revisiones sistemáticas [5]. Además, estos estudios se quedan anticuados muy pronto debido a que no suelen testear las herramientas de forma práctica, descargarlas, instalarlas y usarlas.

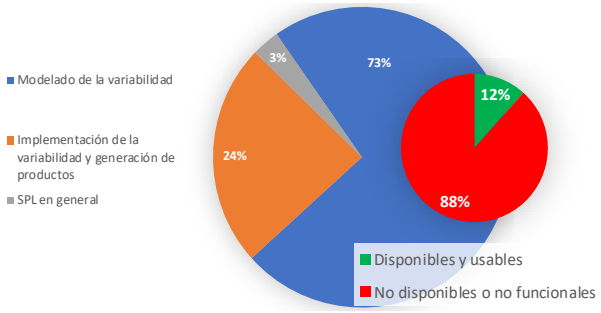

Figura 2. Clasificación de herramientas de SPL.

Esto implica que la mayoría de las herramientas analizadas en esos estudios no están ni siquiera disponibles [34,37].

Otro problema de las revisiones sistemáticas existentes es que su análisis se queda en un nivel muy alto, indicando solo si una determinada fase de las SPLs está cubierta o no, pero sin entrar en los detalles sobre como se proporcionan las características específicas y si es la forma apropiada [37]. La conclusión es que este tipo de estudios no son suficientes para analizar las dificultades de la industria para adoptar un enfoque SPL [5].

De estos estudios [5,34,37] se extrae que el 73 % de las herramientas de SPL (de un total de 97) se enfocan en modelar la variabilidad. A partir de estas herramientas se ha realizado un estudio empírico descargando y probando las herramientas, que ha dado como resultado que solo el 12 % están disponibles para descargar y/o son utilizables (Figura 2) — i.e., son versiones estables que funcionan correctamente sin errores. En la siguiente sección se tienen en cuenta las seis herramientas más relevantes [5].

3. Modelado de la variabilidad en SPL complejas

Esta sección presenta los aspectos de modelado de la variabilidad más complejos que aparecen en los proyectos actuales de SPL y que no están debidamente cubiertos por las herramientas existentes, lo que muchas veces impide la adoptación de la SPL en las empresas. Para cada aspecto o característica, describimos brevemente en qué consiste, damos un ejemplo de aplicación o domino en el cual se usa o requiere de dicha característica de modelado, presentamos el soporte que dan las herramientas existentes, y discutimos como debería ser el soporte ideal para dicha característica. La Tabla 1 resume el soporte actual de las herramientas existentes para cada uno de los aspectos descritos. Se han escogido esas herramientas porque son las más relevantes, están disponibles para descargar y tiene buena usabilidad [5,34,37].

3.1. Modelado de la variabilidad

El soporte más básico para modelar la variabilidad en cualquier dominio de aplicación incluye la posibilidad de definir características opcionales (*optional features*), obligatorias (*mandatory features*), grupos de multiplicidad (*or* y *xor*), y restricciones simples (del tipo *requires* y *excludes*). Prácticamente, todas las herramientas de modelado existentes soportan estas características básicas como muestra la Tabla 1. En esta sección nos centramos en el modelado de aquellas características de variabilidad más complejas.

Tabla 1. Soporte práctico de las herramientas de modelado de la variabilidad en SPLs.

Características	S.P.L.O.T.[1]	Glencoe[2]	Clafer[3]	FAMA[4]	FeatureIDE[5]	pure::variants[6]
Modelado de la variabilidad						
Modelado básico de la variabilidad - Características opcionales, obligatorias, grupos de características (*or*, *xor*), restricciones (*requires*, *excludes*).	■	■	■	■	■	■
Modelo de características extendidos - Características con atributos.	□	□	◪	■	◪	■
Características con tipo variable - Características numéricas (*integer*, *float*), cadenas de caracteres (*string*), etc.	□	□	■	◪	□	◪
Características con cardinalidad - Características clonables.	□	□	◪	□	◪	◪
Restricciones complejas y otras extensiones - Lógica de segundo orden, lógica modal, lenguajes de restricciones (*OCL*), características abstractas, grupos de multiplicidad arbitrarios, etc.	□	□	■	◪	◪	■
Análisis de la variabilidad						
Análisis automático de modelos de variabilidad - Validación, estadísticas y métricas, operaciones de análisis (características muertas, anomalías,...).	◪	◪	□	■	◪	◪
Generación de configuraciones y proceso de configuración - Número de configuraciones, enumeración de configuraciones, optimización de configuraciones, configuración parciales, configuración por pasos, etc.	◪	□	◪	◪	◪	◪
Gestión de los modelos						
Composición de características y múltiples puntos de vista - Unidades de modularización (*composite units*), y *viewpoints*: objetivos, contexto, atributos de calidad, despliegue, tecnologías, funcionalidad, etc.	□	□	◪	□	◪	◪
Formato de los modelos y compatibilidad entre herramientas - Estándarización, formato intercambiables, importación/exportación, etc.	□	■	□	◪	◪	◪
Evolución de los modelos - Modificación de características (e.g., cambiar el tipo de variabilidad de una características, moverla a otra parte del modelo, etc.).	□	□	◪	□	◪	◪
Trazabilidad e independencia de la variabilidad - Independencia y relación entre el modelo de variabilidad y los artefactos.	□	□	□	□	◪	◪

■ Soporte completo. ◪ Soporte parcial. □ Sin soporte.
[1] S.P.L.O.T. [36]: `http://www.splot-research.org/`
[2] Glencoe [1]: `https://glencoe.hochschule-trier.de/`
[3] Clafer [2]: `https://www.clafer.org/`
[4] FAMA [9]: `https://www.isa.us.es/fama/`
[5] FeatureIDE [33,49]: `http://www.featureide.com/`
[6] pure::variants [46]: `https://www.pure-systems.com/`

Modelando propiedades no funcionales

Descripción: Benavides et al. [11] introdujo los **modelos de características extendidos** que permiten definir **atributos** sobre las características del modelo de variabilidad. Estos atributos suelen usarse para modelar propiedades no funcionales del sistema como el coste de una característica, su rendimiento, latencia, consumo de memoria, etc.

Dominio de aplicación: Aquellas aplicaciones que necesiten desplegar configuraciones acorde a ciertas propiedades no funcionales del sistema y optimizar configuraciones. Por ejemplo, las aplicaciones móviles pueden desplegarse con configuraciones que minimicen el consumo de batería según el contexto del dispositivo en el que se ejecuten.

Soporte práctico: Ni S.P.L.O.T. ni Glencoe soportan características con atributos. Clafer permite modelar atributos como características adicionales con tipo variable (e.g.,

variables enteras o reales). Esto quiere decir que para cada característica del árbol de variabilidad debemos añadir una nueva variable que defina ese atributo (e.g., coste) y que funciona como cualquier otra característica del árbol, debiendo seleccionarse dicha variable como una características más cuando se generen configuraciones del modelo. Esta solución es muy poco práctica porque incrementa de forma innecesaria el grado de variabilidad del modelo. FeatureIDE soporte los atributos solo parcialmente, ya que requiere usar un tipo especial de extensión o *composer* que no es compatible con el resto de plugins para implementar la variabilidad, por lo que solo permite especificar los atributos a nivel del árbol de variabilidad, además de tener que especificarlos directamente en el XML del modelo sin poder usar el editor gráfico para ello. Por el contrario, pure::variants y FAMA ofrecen soporte completo para los atributos.

Discusión: Los atributos son propiedades de las características del modelo de variabilidad por lo que tienen que ser definidas como tal y no como características propias (*features*) en si mismas [11] (como ocurre con Clafer). Los atributos pueden ser considerados a la hora de generar configuraciones basadas en sus valores (e.g., propiedades no funcionales) para optimizar las configuraciones. Sin embargo, valores diferentes de un atributo asociado a una característica no supone una configuración diferente del modelo porque un atributo no es un punto de variación. Por otro lado, existen enfoques específicos y más adecuados para modelar la variabilidad de propiedades no funcionales como el framework NFR [14], y que sería deseable integrar en una herramienta de SPL.

Variabilidad de grano fino: parámetros variables

Descripción: Una **característica con tipo variable** [15] define un punto de variación en el cual hay que proporcionar un valor (e.g., entero, real, cadena de caracteres, fecha) para resolver esa variabilidad. Es importante diferenciar entre características con tipo variable, que son aquellas que requieren proporcionar un valor cuando se genere una configuración del modelo, y características con atributos [11], que son usadas para modelar propiedades no funcionales de las características.

Dominio de aplicación: En muchas aplicaciones es necesario modelar la variabilidad de parámetros configurables o variables del sistema (e.g., tamaño del mensaje en kB, número de píxeles en una imagen, etc). En estos casos la forma adecuada de modelarlos es usando características con tipo variable.

Soporte práctico: Clafer es la herramienta con mejor soporte para características con tipo variable. Clafer permite definir variables con un tipo específico (e.g., enteros o reales), así como definir restricciones sobre los valores de dichas variables. El resto de herramientas no soportan este tipo de características, aunque en pure::variants pueden modelarse como si fueran atributos.

Discusión: El soporte para características con tipo variable es confuso debido a la sutil diferencia que existe con las características con atributos. Hay que tener en cuenta que un valor diferente en una característica con tipo variable supone una configuración diferente del modelo de variabilidad, mientras que un valor diferente de un atributo asociado a una característica no supone una configuración diferente (aunque actualmente no hay consenso en la comunidad sobre esto). Por otro lado, si el rango de valores que puede tener una variable no es muy elevado, se puede discretizar y modelarse como características normales que representan cada uno de los valores. Si por el contrario el rango de valores es muy amplio, sería deseable poder definir intervalos de valores.

Modelando múltiples instancias de una misma característica
Descripción: Czarnecki et al. [19] introdujo el concepto de modelos de **características con cardinalidad (clonables)**, donde a una característica se le puede asociar una cardinalidad (e.g., [1..*]) indicando que puede ser instanciada múltiples veces. Esto permite configurar el subárbol correspondiente de esa característica de forma independiente para cada instancia.

Dominio de aplicación: En sistemas IoT [4] y CPS [26], las características clonables son esenciales para modelar la variabilidad de los diferentes dispositivos, tanto a nivel hardware como software. Por ejemplo, una aplicación *multi-tenant* [25] necesitará modelar la variabilidad de los *tenants* como una característica clonable para posteriormente configurar cada instancia (*tenant*) de forma diferente. De igual modo, en un sistema multi-agente de una casa automatizada [3], cada agente que controla la funcionalidad de un electrodoméstico debe modelarse como una instancia de la característica clonable.

Soporte práctico: Las características clonables es uno de los aspectos más difícil de implementar, y por lo tanto ninguna herramienta proporcionar un soporte completo para ellas. Clafer permite clonar cualquier características del modelo y configurar cada instancia, pero tiene que hacerse durante la fase de configuración del modelo y no durante el modelado de la variabilidad que es donde realmente se debería definir si una característica es clonable o no. FeatureIDE y pure::variants permiten un comportamiento similar a las clonables por medio de la incorporación de subárboles en el modelo de variabilidad. Sin embargo, esto obliga a decidir el número de instancias durante el modelado, cosa que se debería decidir durante la fase de configuración.

Discusión: La cardinalidad de una característica clonable debe definirse en la ingeniería del dominio de la SPL, mientras que el número de instancias clonables para una configuración del modelo debe especificarse en la ingeniería de la aplicación. A su vez, es deseable especificar restricciones entre diferentes clones y para ello se necesita un soporte de restricciones más completo, como se discute en el siguiente punto.

Modelando dependencias y otros aspectos
Descripción: La incorporación de características clonables y características con tipo variable conllevan la posible inclusión de **restricciones complejas**. Por ejemplo, el modelado de dependencias entre características que son hijas de diferentes características clonables [19], o dependencias que tengan en cuenta valores numéricos.

Dominio de aplicación: Cualquier aplicación o dominio que requiera de características no básicas como las clonables o las variables numéricas necesitará definir restricciones sobre ellas.

Soporte práctico: Solo Clafer (con restricciones que pueden incluir variables numéricas) y pure::variants (con Prolog o su propia variante de OCL: pvSCL [46]) permiten definir restricciones complejas. El resto de herramientas solo permiten definir restricciones con lógica de primer orden (*and*, *or*, *not*, *implies*).

Discusión: Las herramientas deberían proporcionar la posibilidad de definir restricciones en lógica de mayor orden (high-order logic constraints), en lenguajes estándares de modelado de restricciones como OCL, lenguajes de programación como Prolog, o incluso definir su propio lenguaje de restricciones si fuera necesario, como ocurre con pure::variants y pvSCL [46]. Estas restricciones se deberían poder aplicar sobre cualquier característica del modelo o incluso sobre diferentes modelos de variabilidad.

3.2. Análisis de la variabilidad

Una vez modelada la variabilidad es importante poder razonar sobre ella [6] y analizar las posibles configuraciones del modelo.

Análisis automático de la variabilidad

Descripción: Benavides et al. [8] presenta una revisión sistemática sobre análisis de la variabilidad y define un conjunto de operaciones sobre los modelos de variabilidad: desde la validación del modelo hasta la generación de configuraciones, pasando por la identificación de anomalías, métricas y propiedades (homogeneidad, grado de variabilidad, ortogonalidad,...), estadísticas (características comunes y variables, conjunto atómicos,...) y optimización de configuraciones. Para ello es necesario poder definir o transformar el modelo de variabilidad a diferentes formalizaciones como satisfacción de restricciones (SAT) [6,35], diagramas de decisión binarios (BDD, de *Binary Decision Diagrams*) [7], o programación con restricciones [10].

Dominio de aplicación: El análisis automático de la variabilidad es deseable en cualquier dominio de aplicación, pero su importancia se incrementa en aquellos dominios donde el número de características es elevado, como en los sistemas IoT.

Soporte práctico: Casi todas las herramientas proporcionan algún tipo de soporte para analizar los modelos de variabilidad. Clafer y FeatureIDE usan resolutores de propagación de restricciones (CPS, de *Constraint Propagation Solver*) [10], mientras que S.P.L.O.T., Glencoe y FAMA usan varios resolutores para soportar diferentes formalizaciones (SAT, BDD) [8,10].

Discusión: Dependiendo del análisis requerido, diferentes formalizaciones del modelo de variabilidad deberían ser soportadas. Por ejemplo, para analizar el grado de variabilidad es mejor utilizar una formalización en BDD [7] que permite contar el número de configuraciones válidas del modelo de manera directa en lugar de tener que generar todas las configuraciones para contarlas como haría un resolutor CPS [10].

Trabajando con configuraciones del modelo

Descripción: Generar una configuración del modelo seleccionando las características requeridas no es trivial, sobre todo si necesitamos optimizar las configuraciones en base a algún criterio (e.g., propiedades no funcionales). Para ello puede ser útil trabajar con configuraciones parciales, realizar el proceso de configuración por pasos (*step-by-step configuration*) o en diferentes modelos (*viewpoints*), o incluso generar todas las posibles configuraciones (si fuera posible) para razonar sobre ellas.

Dominio de aplicación: Aplicaciones que requieren generar configuraciones en base a objetivos o criterios. Por ejemplo, encontrar la configuración que minimiza el consumo energético de la aplicación [26].

Soporte práctico: Sólo S.P.L.O.T. (con un asistente para configuración por pasos) y Clafer (con un proceso de instanciación basado en restricciones) proporcionan un soporte razonable para configuraciones parciales y configuración por pasos. FeatureIDE y pure::variants permiten generar configuraciones parciales pero no incorporan ningún proceso guiado para asistir al usuario. Por otro lado, solo Clafer y FeatureIDE permiten generar y enumerar todas las configuraciones válidas, aunque esto no es factible para modelos muy grandes.

Discusión: Hoy en día, ninguna herramienta permite generar todas las configuraciones de forma eficiente para modelos muy grandes. Es más, aunque las herramientas permiten calcular el número de configuraciones, no se tiene en cuenta las características numéricas en ese cálculo. Por otro lado, sería deseable asistir al usuario a la hora de generar una configuración del modelo de variabilidad informando en todo momento de las diferentes posibilidades (e.g., configuraciones óptimas, configuraciones inválidas,...). De hecho, no existe ninguna herramienta operativa que permita generar configuraciones óptimas, por ejemplo, basadas en propiedades no funcionales del sistema [25].

3.3. Gestión de los modelos de variabilidad

En esta sección se discute la gestión y el tratamiento de los modelos de variabilidad, como su tamaño, los formatos, la evolución de los mismos y su conexión con los artefactos que implementan la variabilidad.

Trabajando con modelos de gran tamaño

Descripción: Los modelos de variabilidad de gran tamaño pueden llegar a ser inmanejables y es necesario dividirlos, usando unidades de modularización como por ejemplo **composición de características** (*composite variability models*) [15], **múltiples puntos de vista** (*viewpoints*), o incluso Líneas de Productos Múltiples (*MultiPLs*) [41]. Por ejemplo, siguiendo un enfoque holístico [30] la variabilidad del sistema se puede dividir en varios modelos interconectados. Por ejemplo, un modelo para especificar la variabilidad de los objetivos del usuario, otro para la variabilidad del contexto, otro para la variabilidad de los atributos de calidad, otro con la variabilidad de la funcionalidad principal del sistema, otro con la variabilidad de las tecnologías disponibles (e.g., lenguajes de programación), y otro con la variabilidad de los dispositivos para el despliegue del sistema [28].

Dominio de aplicación: Los sistemas de hoy en día disponen fácilmente de más de un centenar de características. Por ejemplo, la SPL del sistema operativo Linux contiene miles de características diferentes [7]. Un modelo con 100 características puede llegar a tener más de 10^{30} posibles configuraciones. Además, el problema del número de configuraciones se incrementa considerablemente si consideramos dominios donde intervienen las características clonables y los parámetros variables.

Soporte práctico: Por un lado, ninguna herramienta ofrece un soporte adecuado para trabajar con modelos de gran tamaño (más de 100 características). Primero, los modelos deben ser definidos a mano, normalmente mediante un editor gráfico (salvo Clafer y FAMA), lo que dificultad trabajar con modelos grandes. Y segundo, dependiendo de la operación que se realice sobre el modelo (análisis, generación de configuraciones,...), la complejidad del problema puede pasar de lineal a exponencial (muchas de estas operaciones son problemas NP-completos).

Por otro lado, los modelos de variabilidad no se pueden modularizar de forma fácil con el soporte de herramientas existentes. Clafer permite definir múltiples modelos de variabilidad como clases abstractas pero todos ellos en el mismo fichero. FeatureIDE soporta MultiPLs [41] pero esta característica es un prototipo a día de hoy con muchos fallos, y además, el modelo de variabilidad en sí no puede dividirse en varios ficheros reutilizables. En pure::variants, el soporte es algo mejor ya que se define una relación de composición que permite definir un enlace entre modelos para incorporar uno dentro

de otro. Sin embargo, ninguna herramienta proporciona soporte para definir múltiples puntos de vista y relacionarlos entre sí.

Discusión: El primer paso para poder trabajar con modelos gigantes es definir un lenguaje de especificación (preferiblemente textual, aunque luego tenga su representación gráfica), que permita definir los modelos de forma sencilla y rápida. A partir de ahí, el soporte para unidades de modularización, composición de características, múltiples puntos de vista, y MultiPLs es indispensable para modelos gigantescos.

Formato de los modelos y compatibilidad entre herramientas

Descripción: Después de casi 30 años de investigación sobre modelos de variabilidad, aún no se ha aceptado ningún formato de modelado como estándar, y las propuestas de estandarización (CVL [15] y EMF [47]) no han tenido éxito. Por lo que cada herramienta ha definido su propio formato y notación propios, lo que resulta en una gran diversidad de formatos (SXFM, GUIDSL, Velvet, DIMACS,...).

Dominio de aplicación: El formato es independiente del dominio de aplicación. Sin embargo, hay sistemas donde intervienen varios dominios con requisitos de variabilidad diferentes, siendo necesario usar varias herramientas de modelado.

Soporte práctico: Mientras que FeatureIDE permite importar los modelos de variabilidad generados con S.P.L.O.T. (en formato SXFM), Glencoe permite exportar su modelo de variabilidad a una gran variedad de formatos (DIMACS, SPASS, v.control,...), incluyendo el formato de pure::variants. Clafer y FAMA usan sus propios formatos y notaciones.

Discusión: Mientras no haya una propuesta integral de una herramienta que incluya todas las características requeridas por los diferentes dominios, será necesario definir formatos intercambiables fáciles de importar y exportar entre las diferentes herramientas.

Evolución de los modelos

Descripción: Cuando los requisitos de las aplicaciones cambian o la tecnología evoluciona, los modelos de variabilidad deben modificarse añadiendo nuevas características o eliminando otras. Además, estos cambios deben propagarse a las configuraciones desplegadas.

Dominio de aplicación: En algunos dominios, como en los sistemas multi-tenant o en sistemas multi-agentes [4,3], con cientos de configuraciones desplegadas, modificar los modelos de variabilidad y propagar los cambios a las configuraciones es una tarea imposible de realizar manualmente.

Soporte práctico: Realizar modificaciones al modelo de variabilidad una vez creado puede ser complejo en herramientas como S.P.L.O.T. y Glencoe, donde modificar una parte del modelo significa eliminar y volver a crear dicha parte. Por el contrario, Clafer y pure::variants permiten mover de sitio ramas completas del árbol de variabilidad de forma trivial. A pesar de ello, ninguna herramienta proporciona soporte para adaptar las configuraciones ya generadas de acuerdo a los cambios en el modelo de variabilidad.

Discusión: El soporte para mantener y evolucionar los modelos de variabilidad y las configuraciones es tan importante como poder definir todas las características discutidas anteriormente, ya que una SPL está en continuo desarrollo y crecimiento, sobre todo si la empresa ha seguido un enfoque de adopción extractivo [8].

Trazabilidad e independencia de la variabilidad

Descripción: Otro punto diferente a tener en cuenta es el formato de los modelos sobre los que se especifica la variabilidad (los modelos de los artefactos variables) y como se relacionan estos con las características definidas en el modelo de variabilidad (i.e., trazabilidad). Idealmente la variabilidad debería poder especificarse sobre cualquier modelo independientemente de su formato (UML, BPMN, MOF,...) o sobre cualquier lenguaje de programación.

Dominio de aplicación: Cada dominio de aplicación tiene sus propios modelos con sus propios formatos y/o lenguajes de programación (e.g., modelos de proceso BPMN, UML, lenguajes de dominio específico, lenguajes de descripción arquitectónicos, código en diferentes lenguajes de programación). Por ejemplo, en las aplicaciones web es común usar diferentes lenguajes de programación y scripting (JavaScript, Java, Python) y lenguajes de marcado (HTML, CSS, XML) donde especificar la variabilidad y conectarlo con el modelo de variabilidad no es trivial.

Soporte práctico: Los modelos de variabilidad de las herramientas suelen ser independientes del modelo sobre el que se aplique la variabilidad. Aunque esta independencia a su vez impide conectar el modelo de variabilidad con los artefactos y sus puntos de variación. En este sentido, solo FeatureIDE y pure::variants permiten conectar el modelo de variabilidad con el código de las aplicaciones, permitiendo usar diferentes lenguajes (Java, C++), pero no combinarlos entre sí. Las otras herramientas no proporcionan ningún soporte para la trazabilidad de la variabilidad.

Discusión: La variabilidad debe poder especificarse sobre cualquier lenguaje de modelado o de programación, preferiblemente siguiente un enfoque ortogonal donde la información de variabilidad no se incluye directamente en el modelo del dominio o en los artefactos, sino que se especifica independientemente en el modelo de variabilidad y mediante un modelo de *mapping* se relacionan (o mapea) con los artefactos variables. Por ejemplo, el lenguaje CVL permite especificar la variabilidad sobre cualquier modelo basado en MOF, sin embargo no existe soporte disponible o usable para el lenguaje CVL a día de hoy.

4. Conclusiones y Trabajo Futuro

Del análisis realizado en la sección anterior se concluye que no existe un enfoque o solución integral que recoja todas las características complejas de variabilidad requeridas en los dominios de aplicación actuales. Es necesario adaptar los requisitos de las SPLs a las funcionalidades proporcionadas por las herramientas, lo que hace que las empresas sean reticentes a adoptar un enfoque de SPL. Con este artículo sentamos las bases para la definición de los artefactos necesarios (herramientas, lenguajes, metamodelos) que den soporte a las características complejas de variabilidad que hoy en día exigen los enfoques de SPL. Como trabajo futuro se plantea analizar el soporte práctico de otros aspectos de las SPLs, más allá del modelado de la variabilidad, como son las técnicas de implementación de la variabilidad, o las Líneas de Productos Software Dinámicas (DSPLs) [23].

Agradecimientos

Trabajo financiado por los proyectos Magic P12-TIC181, HADAS TIN2015-64841-R (cofinanciado con fondos FEDER), TASOVA MCIU-AEI. REF: TIN2017-90644-REDT, y MEDEA RTI2018-099213-B-I00.

Referencias

1. Anna Schmitt, Christian Bettinger, G.R.: Glencoe – a tool for specification, visualization and formal analysis of product lines. In: Transdisciplinary Engineering Methods for Social Innovation of Industry 4.0. Advances in Transdisciplinary Engineering, vol. 7, pp. 665–673 (2018). https://doi.org/10.3233/978-1-61499-898-3-665
2. Antkiewicz, M., Bąk, K., Murashkin, A., Olaechea, R., Liang, J.H.J., Czarnecki, K.: Clafer tools for product line engineering. In: International Software Product Line Conference Co-located Workshops. pp. 130–135. SPLC (2013). https://doi.org/10.1145/2499777.2499779
3. Ayala, I., Amor, M., Horcas, J.M., Fuentes, L.: Model driven evolution of an agent-based home energy management system. In: New Trends in Intelligent Software Methodologies, Tools and Techniques. pp. 17–30. SoMeT (2018). https://doi.org/10.3233/978-1-61499-900-3-17
4. Ayala, I., Horcas, J.M., Amor, M., Fuentes, L.: Using models at runtime to adapt self-managed agents for the iot. In: German Conference on Multiagent System Technologies. pp. 155–173. MATES (2016). https://doi.org/10.1007/978-3-319-45889-2_12
5. Bashroush, R., Garba, M., Rabiser, R., Groher, I., Botterweck, G.: CASE tool support for variability management in software product lines. ACM Comput. Surv. **50**(1), 14:1–14:45 (2017). https://doi.org/10.1145/3034827
6. Batory, D.S.: Feature models, grammars, and propositional formulas. In: Software Product Lines, 9th International Conference, SPLC 2005, Rennes, France, September 26-29, 2005, Proceedings. pp. 7–20 (2005). https://doi.org/10.1007/11554844_3, `https://doi.org/10.1007/11554844\_3`
7. Batory, D.S., Benavides, D., Cortés, A.R.: Automated analysis of feature models: challenges ahead. Commun. ACM **49**(12), 45–47 (2006). https://doi.org/10.1145/1183236.1183264, `https://doi.org/10.1145/1183236.1183264`
8. Benavides, D., Segura, S., Ruiz-Cortés, A.: Automated analysis of feature models 20 years later: A literature review. Information Systems **35**(6), 615 – 636 (2010). https://doi.org/https://doi.org/10.1016/j.is.2010.01.001
9. Benavides, D., Segura, S., Trinidad, P., Cortés, A.R.: FAMA: tooling a framework for the automated analysis of feature models. In: International Workshop on Variability Modelling of Software-Intensive Systems. pp. 129–134. VaMoS (2007)
10. Benavides, D., Trinidad, P., Cortés, A.R.: Using constraint programming to reason on feature models. In: International Conference on Software Engineering and Knowledge Engineering. pp. 677–682. SEKE (2005)
11. Benavides, D., Trinidad, P., Cortés, A.R.: Automated reasoning on feature models. In: Advanced Information Systems Engineering. pp. 491–503. CAiSE (2005). https://doi.org/10.1007/11431855_34
12. Bosch, J., Florijn, G., Greefhorst, D., Kuusela, J., Obbink, J.H., Pohl, K.: Variability issues in software product lines. In: Software Product-Family Engineering. pp. 13–21. PFE (2001). https://doi.org/10.1007/3-540-47833-7_3
13. Capilla, R., Dueñas, J.C.: Modelling variability with features in distributed architectures. In: Software Product-Family Engineering. pp. 319–329. PFE (2001). https://doi.org/10.1007/3-540-47833-7_28
14. Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-functional requirements in software engineering, vol. 5. Springer Science & Business Media (2012)
15. CVL Submission Team: Common variability language (CVL), OMG revised submission. http://www.omgwiki.org/variability/lib/exe/fetch.php?id=start&cache=cache&media=cvl- revised- submission.pdf. (2012)
16. Czarnecki, K., Bednasch, T., Unger, P., Eisenecker, U.W.: Generative programming for embedded software: An industrial experience report. In: Generative Programming and Component Engineering. pp. 156–172. GPCE (2002). https://doi.org/10.1007/3-540-45821-2_10
17. Czarnecki, K., Eisenecker, U.W.: Generative programming - methods, tools and applications. Addison-Wesley (2000)

18. Czarnecki, K., Helsen, S., Eisenecker, U.W.: Staged configuration using feature models. In: International Conference on Software Product Lines. pp. 266–283. SPLC (2004). https://doi.org/10.1007/978-3-540-28630-1_17
19. Czarnecki, K., Helsen, S., Eisenecker, U.W.: Formalizing cardinality-based feature models and their specialization. Software Process: Improvement and Practice **10**(1), 7–29 (2005). https://doi.org/10.1002/spip.213
20. Czarnecki, K., She, S., Wasowski, A.: Sample spaces and feature models: There and back again. In: International Conference on Software Product Lines. pp. 22–31. SPLC (2008). https://doi.org/10.1109/SPLC.2008.49
21. Ferber, S., Haag, J., Savolainen, J.: Feature interaction and dependencies: Modeling features for reengineering a legacy product line. In: International Conference Software Product Lines. pp. 235–256. SPLC (2002). https://doi.org/10.1007/3-540-45652-X_15
22. Griss, M.L., Favaro, J.M., D'Alessandro, M.: Integrating feature modeling with the RSEB. In: International Conference on Software Reuse. pp. 76–85. ICSR (1998). https://doi.org/10.1109/ICSR.1998.685732
23. Hallsteinsen, S.O., Hinchey, M., Park, S., Schmid, K.: Dynamic software product lines. IEEE Computer **41**(4), 93–95 (2008). https://doi.org/10.1109/MC.2008.123
24. Haugen, Ø., Øgård, O.: BVR - better variability results. In: International Conference on System Analysis and Modeling: Models and Reusability. pp. 1–15. SAM (2014). https://doi.org/10.1007/978-3-319-11743-0_1
25. Horcas, J.M.: WeaFQAs: A Software Product Line Approach for Customizing and Weaving Efficient Functional Quality Attributes. phdthesis, Universidad de Málaga (2018), `https://hdl.handle.net/10630/17231`
26. Horcas, J.M., Pinto, M., Fuentes, L.: Context-aware energy-efficient applications for cyber-physical systems. Ad Hoc Networks **82**, 15–30 (2019). https://doi.org/10.1016/j.adhoc.2018.08.004
27. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (foda) feasibility study. Tech. rep., Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst (1990)
28. Kang, K.C., Lee, H.: Variability Modeling, pp. 25–42. Springer Berlin Heidelberg (2013). https://doi.org/10.1007/978-3-642-36583-6_2
29. Kang, K.C., Kim, S., Lee, J., Kim, K., Shin, E., Huh, M.: FORM: A feature-oriented reuse method with domain-specific reference architectures. Ann. Software Eng. **5**, 143–168 (1998). https://doi.org/10.1023/A:1018980625587
30. Lee, J., Kang, K.C., Sawyer, P., Lee, H.: A holistic approach to feature modeling for product line requirements engineering. Requirements Engineering **19**(4), 377–395 (2014). https://doi.org/10.1007/s00766-013-0183-6
31. Lee, J., Kang, K.C.: Feature binding analysis for product line component development. In: International Workshop on Software Product-Family Engineering. pp. 250–260. PFE (2003). https://doi.org/10.1007/978-3-540-24667-1_18
32. Lee, K., Kang, K.C.: Usage context as key driver for feature selection. In: International Conference on Software Product Lines. pp. 32–46. SPLC (2010). https://doi.org/10.1007/978-3-642-15579-6_3
33. Leich, T., Apel, S., Marnitz, L., Saake, G.: Tool support for feature-oriented software development: Featureide: an eclipse-based approach. In: Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology eXchange. pp. 55–59. eclipse (2005). https://doi.org/10.1145/1117696.1117708
34. Lisboa, L.B., Garcia, V.C., Lucrédio, D., de Almeida, E.S., de Lemos Meira, S.R., de Mattos Fortes, R.P.: A systematic review of domain analysis tools. Information and Software Technology **52**(1), 1–13 (2010). https://doi.org/https://doi.org/10.1016/j.infsof.2009.05.001
35. Mannion, M.: Using first-order logic for product line model validation. In: International Conference on Software Product Lines. pp. 176–187. SPLC (2002). https://doi.org/10.1007/3-540-45652-X_11

36. Mendonca, M., Branco, M., Cowan, D.: S.p.l.o.t.: Software product lines online tools. In: ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications. pp. 761–762. OOPSLA (2009). https://doi.org/10.1145/1639950.1640002
37. Pereira, J.A., Constantino, K., Figueiredo, E.: A systematic literature review of software product line management tools. In: Software Reuse for Dynamic Systems in the Cloud and Beyond. pp. 73–89 (2014)
38. Pohl, K., Böckle, G., Linden, F.J.v.d.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag (2005)
39. Raatikainen, M., Tiihonen, J., Männistö, T.: Software product lines and variability modeling: A tertiary study. Journal of Systems and Software **149**, 485–510 (2019). https://doi.org/https://doi.org/10.1016/j.jss.2018.12.027
40. Rabiser, R., Schmid, K., Becker, M., Botterweck, G., Galster, M., Groher, I., Weyns, D.: A study and comparison of industrial vs. academic software product line research published at SPLC. In: International Conference on Systems and Software Product Line. pp. 14–24. SPLC (2018). https://doi.org/10.1145/3233027.3233028
41. Rosenmüller, M., Siegmund, N., Thüm, T., Saake, G.: Multi-dimensional variability modeling. In: Variability Modeling of Software-Intensive Systems. pp. 11–20. VaMoS (2011). https://doi.org/10.1145/1944892.1944894
42. Schmid, K., John, I.: A customizable approach to full lifecycle variability management. Science of Computer Programming **53**(3), 259–284 (2004). https://doi.org/https://doi.org/10.1016/j.scico.2003.04.002
43. Schobbens, P., Heymans, P., Trigaux, J., Bontemps, Y.: Generic semantics of feature diagrams. Computer Networks **51**(2), 456–479 (2007). https://doi.org/10.1016/j.comnet.2006.08.008
44. Sinnema, M., Deelstra, S., Nijhuis, J., Bosch, J.: COVAMOF: A framework for modeling variability in software product families. In: International Conference on Software Product Lines. pp. 197–213. SPLC (2004). https://doi.org/10.1007/978-3-540-28630-1_12
45. Sousa, G., Rudametkin, W., Duchien, L.: Extending dynamic software product lines with temporal constraints. In: International Symposium on Software Engineering for Adaptive and Self-Managing Systems. pp. 129–139. SEAMS (2017). https://doi.org/10.1109/SEAMS.2017.6
46. Spinczyk, O., Beuche, D.: Modeling and building software product lines with eclipse. In: ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications. pp. 18–19. OOPSLA (2004). https://doi.org/10.1145/1028664.1028675
47. Steinberg, D., Budinsky, F., Merks, E., Paternostro, M.: EMF: eclipse modeling framework. Pearson Education (2008)
48. Thüm, T., Batory, D.S., Kästner, C.: Reasoning about edits to feature models. In: International Conference on Software Engineering. pp. 254–264. ICSE (2009). https://doi.org/10.1109/ICSE.2009.5070526
49. Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., Leich, T.: Featureide: An extensible framework for feature-oriented software development. Science of Computer Programming **79**, 70–85 (2014). https://doi.org/https://doi.org/10.1016/j.scico.2012.06.002
50. White, J., Dougherty, B., Schmidt, D.C., Benavides, D.: Automated reasoning for multi-step feature model configuration problems. In: International Software Product Line Conference. pp. 11–20. SPLC, Carnegie Mellon University (2009)
51. Zhang, G., Ye, H., Lin, Y.: Quality attribute modeling and quality aware product configuration in software product lines. Software Quality Journal **22**(3), 365–401 (2014). https://doi.org/10.1007/s11219-013-9197-z
52. Ø. Haugen, Møller-Pedersen, B., Oldevik, J., Olsen, G.K., Svendsen, A.: Adding standardized variability to domain specific languages. In: International Software Product Line Conference. pp. 139–148. SPLC (2008). https://doi.org/10.1109/SPLC.2008.25