vEXgine: Extendiendo el Motor de Ejecución de CVL

José Miguel Horcas, Mónica Pinto y Lidia Fuentes

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

Resumen El Lenguaje CVL (*Common Variability Language*) carece de una herramienta flexible que permita poner en práctica las necesidades industriales del modelado de la variabilidad en Líneas de Producto Software. Las herramientas existentes que proporcionan soporte para CVL son prototipos incompletos, o se centran principalmente en la especificación de la variabilidad, sin llegar a resolverla sobre modelos reales. Además, no existe una API que permita la interacción directa con el motor CVL para extenderlo o usarlo en una aplicación independiente. Este artículo presenta vEXgine, una implementación adaptable y extensible del motor de ejecución de la variabilidad de CVL.
Keywords: CVL, Línea de Producto Software, Variabilidad, vEXgine

1. Introducción

El lenguaje CVL (*Common Variability Language*) [3] ha desplazado a los tradicionales modelos de características (*feature models*) a la hora de gestionar la variabilidad en Líneas de Producto Software (*SPLs*), especialmente en aquellos enfoques centrados en la arquitectura [5]. Esto se debe a la multitud de ventajas que CVL tiene sobre otros lenguajes de modelado de la variabilidad. Por un lado, CVL permite la resolución de variabilidad a nivel de arquitectura software, es compatible con cualquier modelo basado en *MOF* (*Meta Object Facility*), proporciona varios niveles de abstracción y unidades de modularización para modelar la variabilidad, y tiene soporte nativo para atributos y características clonables. Por otro lado, CVL permite conectar el modelo de variabilidad directamente con la arquitectura software mediante la definición de unos puntos de variación que indican cómo se resuelve la variabilidad en la arquitectura software. Durante la resolución de la variabilidad, el motor de ejecución de CVL delega su control en un mecanismo de transformaciones Modelo a Modelo (M2M) a cargo de ejecutar la semántica definida por los puntos de variación. Por ejemplo, eliminar un componente de la arquitectura si no está seleccionado en una configuración del modelo de variabilidad.

A pesar de estas ventajas, muchas propuestas [2,4,5,7] requieren extender o modificar CVL de diferentes formas para satisfacer sus necesidades. Desde modificar el metamodelo de CVL para incorporar nuevos tipos de características al modelo de variabilidad [4], hasta cambiar la semántica de los puntos de variación para aplicar transformaciones de modelo más complejas [5] o aplicar la

Tabla 1. Comparativa de herramientas CVL.

Característica	MoSIS CVL[i]	CVL 2[ii]	BVR[iii]	KCVL[iv]	**vEXgine**[v]
Editor gráfico de modelos de variabilidad	■	■	■	□	□
Restricciones OCL	■	□	■	■	■
Resolución de la variabilidad	■	□	■	■	■
Transformaciones M2M complejas (OVPs)	□	□	□	□	■
Disponibilidad del metamodelo CVL	■	■	■	□	■
Interfaz de programación (API)	□	□	□	□	■

■ Proporciona la característica. □ No proporciona la característica.
[i] `http://www.omgwiki.org/variability/doku.php?id=cvl_tool_from_sintef`
[ii] `http://modelbased.net/tools/cvl-2-tool/`
[iii] `http://modelbased.net/tools/bvr-tool/`
[iv] `https://diverse-project.github.io/kcvl/`
[v] `http://caosd.lcc.uma.es/vexgine/`

variabilidad directamente sobre código [2]. Incluso, algunas propuestas necesitan modificar el motor de ejecución de CVL para incluir pasos adicionales antes de resolver la variabilidad, como incorporar procesos de chequeo de modelos o identificar los puntos de variación en el modelo arquitectónico [7].

Sin embargo, la comunidad de CVL echa en falta herramientas que den soporte completo al lenguaje (Tabla 1). En particular, ninguna de las herramientas actuales permite extender el enfoque de CVL, cambiar la semántica de los puntos de variación predefinidos, ni definir nuevos puntos de variación con semánticas personalizadas — i.e., definir *OVPs* (*Opaque Variation Points*). Tampoco existe una API para CVL que permita la interacción directa con su motor de ejecución para usarlo en una aplicación independiente o que permita su extensión.

Este artículo presenta *vEXgine* [6][1], una implementación real del motor de ejecución de CVL que permite extender el proceso de resolución de la variabilidad para usar diferentes mecanismos de transformación de modelos. En particular, se proporciona una implementación del mecanismo de transformación M2M sobre el lenguaje de transformación *ATL* (*ATL Transformation Language*).

2. vEXgine

La herramienta *vEXgine*[2] es una implementación del motor de ejecución de CVL para resolver la variabilidad (Figura 1). Las principales características de vEXgine son: (1) soporta completamente el proceso de materialización (esto es, el proceso de resolución de la variabilidad), incluyendo el mecanismo de delegación; (2) permite definir puntos de variación con semánticas definidas por el usuario (OVPs); (3) puede extenderse con diferentes motores de ejecución para resolver la variabilidad, y en particular, se proporciona una implementación de un motor de ejecución basado en transformaciones M2M sobre el lenguaje ATL y metamodelos UML (Figura 2); y (4) proporciona una API en Java que permite extender el enfoque CVL para satisfacer las necesidades de modelado y resolución de la variabilidad en SPLs, incluso en tiempo de ejecución.

En la Figura 1 se identifican los puntos donde vEXgine extiende CVL de forma consistente, es decir, respetando el estándar CVL [1] para especificar y resolver la variabilidad. Básicamente hay tres puntos de extensión: (1) extender

[1] vEXgine recibió el premio a la mejor herramienta en la conferencia internacional SPLC (*Systems and Software Product Line Conference*) en 2017.
[2] vEXgine está disponible en `http://caosd.lcc.uma.es/vexgine/`.

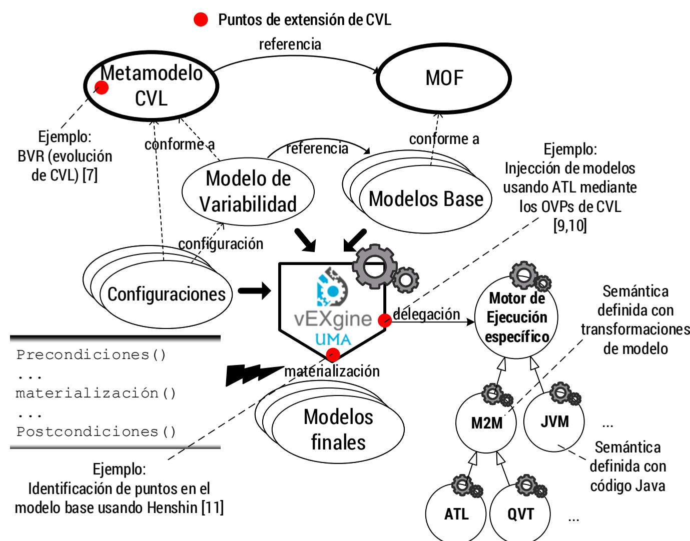

Figura 1. vEXgine y los puntos de extensión de CVL.

el metamodelo de CVL; (2) extender el proceso de materialización; y (3) extender el motor de delegación.

Extensión del metamodelo de CVL. El metamodelo de CVL se puede modificar para extender los modelos de variabilidad y los modelos de configuración. Esto permite definir nuevos constructores para la especificación de relaciones de variabilidad más complejas, y añadir nuevos tipos de puntos de variación. Un ejemplo de este tipo de extensión es *BVR* [4] done se añaden constructores nuevos a cVL para mejorar la expresividad del modelo de variabilidad como características que involucran literales u objetivos.

Con vEXgine se proporciona un metamodelo de CVL extendido para poder definir transformaciones de modelo más complejas (OVPs) que las predefinidas por los puntos de variación de CVL.

Extensión del proceso de materialización. Extender el proceso de materialización de CVL significa modificar el motor de CVL para adaptar o extender el proceso de resolución de la variabilidad a las necesidades de los ingenieros de SPLs. Como muestra la Figura 1, es posible incluir nuevos pasos antes y/o después de la resolución de la variabilidad. Por ejemplo, para validar que la variabilidad se ha resuelto correctamente en el modelo/producto final, o para incluir un proceso previo que identifique de manera segura los puntos donde se va a aplicar la variabilidad en el modelo arquitectónico, como en [7].

Extensión del motor de delegación. Durante su ejecución, CVL delega el control en un motor de transformación M2M encargado de ejecutar las transformaciones definidas por los puntos de variación. Extendiendo este motor de delegación es posible usar nuevos mecanismos para resuelva la variabilidad y nuevos lenguajes de transformación de modelos como *ATL* [5], o *QVT* (*Query/View/Transformation*), o incluso resolver la variabilidad directamente sobre código [2].

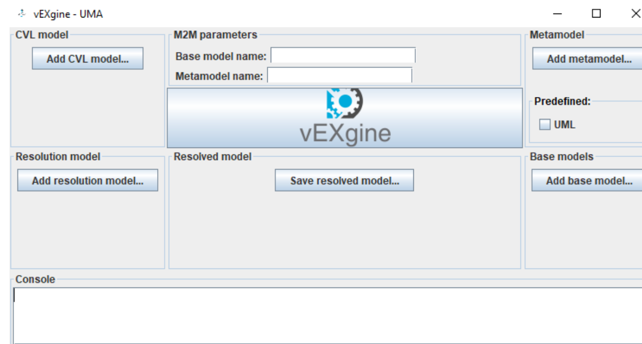

Figura 2. Interfaz gráfica de vEXgine.

3. Conclusiones

La herramienta vEXgine (Figura 2) es una implementación del motor de ejecución de CVL para resolver la variabilidad usando transformaciones de modelo con semánticas personalizadas. La API proporcionada permite usar vEXgine en cualquier aplicación que necesite resolver la variabilidad de modelos CVL. En comparación con las herramientas existentes de CVL, vEXgine puede extenderse para satisfacer las necesidades de los arquitectos software y desarrolladores a través de los puntos de extensión identificados en el enfoque CVL.

Agradecimientos

Trabajo financiado por los proyectos MAGIC P12-TIC1814 y HADAS TIN2015-64841-R, y por la Universidad de Málaga.

Referencias

1. CVL Submission Team: Common Variability Language (CVL), OMG revised submission. http://www.omgwiki.org/variability/ (2012)
2. Filho, J.a.B.F., Allier, S., Barais, O., Acher, M., Baudry, B.: Assessing product line derivation operators applied to java source code: An empirical study. In: International Conference on Software Product Line. pp. 36–45. SPLC (2015)
3. Haugen, Ø., 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)
4. Haugen, Ø., Øgård, O.: BVR — better variability results. In: System Analysis and Modeling: Models and Reusability, pp. 1–15. Springer (2014)
5. Horcas, J.M., Pinto, M., Fuentes, L.: An automatic process for weaving functional quality attributes using a software product line approach. Journal of Systems and Software 112, 78–95 (2016)
6. Horcas, J.M., Pinto, M., Fuentes, L.: Extending the common variability language (CVL) engine: A practical tool. In: Proceedings of the 21st International Systems and Software Product Line Conference, SPLC, Volume B, Sevilla, Spain, September 25-29. pp. 32–37 (2017), http://doi.acm.org/10.1145/3109729.3109749
7. Horcas, J.M., Pinto, M., Fuentes, L., Zschaler, S.: Towards contractual interfaces for reusable functional quality attribute operationalisations. In: International Conference on Modularity. pp. 201–205 (2016)