View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Repositorio Institucional Universidad de Málaga

Reconfigurando Aplicaciones Multi-Cloud con Líneas de Producto Software Dinámicas

Javier Cubo, Nadia Gámez, Ernesto Pimentel, Lidia Fuentes

Dpto de Lenguajes y Ciencias de la Computación, Universidad de Málaga
{cubo, nadia, ernesto, lff}@lcc.uma.es

Resumen. La reconfiguración dinámica de aplicaciones *multi-cloud* es un reto complejo aún no suficientemente explorado. En estos entornos las aplicaciones o sus módulos pueden estar desplegados en diferentes proveedores. Por lo tanto, reconfigurar en tiempo de ejecución estas aplicaciones puede requerir la modificación de la distribución en múltiples y heterogéneos proveedores. Obtener la nueva distribución para que sigan funcionando correctamente las aplicaciones no es una tarea sencilla, pues tanto los requisitos de las aplicaciones como las propiedades de los proveedores son muy diversos y variables. Además, la migración de las aplicaciones o sus módulos en tiempo real de un proveedor a otro puede conllevar problemas de compatibilidad y/o dependencias entre los módulos. Por lo tanto, el manejo de la variabilidad dinámica de las aplicaciones y proveedores, así como el de las dependencias existentes es deseable que se haga a un alto nivel de abstracción. Las Líneas de Producto Software Dinámicas (*DSLP*) utilizan modelos de variabilidad en tiempo de ejecución para obtener los cambios que han de llevarse a cabo durante la reconfiguración. En este trabajo de reflexión, exploramos el uso del enfoque de *DSPL*, para que cuando ocurran problemas en los proveedores o se violen los requisitos de las aplicaciones en entornos *multi-cloud*, las aplicaciones puedan ser reconfiguradas y seguir proporcionando los servicios adecuadamente a los usuarios.

Palabras claves: Reconfiguración; Aplicaciones Cloud; Multi-Cloud; DSPL; Variabilidad.

1 Motivación

En el paradigma de *Cloud Computing* existe una cierta variabilidad en los servicios, requisitos, propiedades y tecnologías soportadas por los diferentes proveedores *cloud*. Para solventar los problemas de portabilidad e interoperabilidad, existen soluciones para desplegar y gestionar las aplicaciones en entornos *multi-cloud*. Estas soluciones soportan el despliegue de aplicaciones o los módulos internos de una aplicación en diferentes proveedores. Una de estas soluciones es la plataforma *SeaClouds (Seamless adaptive multi-cloud management of service-based applications)*[1], desarrollada en el ámbito de un proyecto Europeo en el que dos de los autores de ese trabajo han participado activamente. Esta plataforma ayuda a los desarrolladores y administradores de aplicaciones *cloud*, haciendo más eficiente el diseño, desarrollo,

[1] SeaClouds European Project: http://www.seaclouds-project.eu

planificación y gestión de las aplicaciones complejas en un entorno heterogéneo y *multi-cloud*. De esta manera, cubre todos los aspectos de modelado, planificación y gestión para poder desplegar una aplicación o sus módulos en diversos proveedores que cumplan los requisitos de la aplicación, solventando las dificultades de portabilidad y de interoperabilidad existentes en los despliegues *multi-cloud*.

Sin embargo, estos entornos no son estáticos y en tiempo de ejecución pueden ocurrir problemas inesperados en los proveedores o se pueden violar ciertos requisitos, por lo que también es necesario soportar la reconfiguración dinámica de dichas aplicaciones. En el caso de estos entornos *multi-cloud*, esta reconfiguración se hace más compleja, ya que puede ser necesaria la migración de las aplicaciones (o de los módulos afectados) entre proveedores *cloud* compatibles. *SeaClouds* soporta la migración de un módulo de una aplicación de un proveedor a otro. Pero esta funcionalidad no está integrada dentro de un proceso de auto-adaptación que reaccione cuando ocurren los problemas en tiempo de ejecución, ya que averiguar qué módulos habría que migrar y dónde no es una tarea trivial. Primero sería necesario planificar de nuevo en qué proveedor debe ir cada módulo y esto puede llevar mucho tiempo si se trata de una aplicación compuesta de muchos módulos. En ciertas situaciones, donde la funcionalidad debe verse interrumpida el mínimo tiempo posible, calcular esta nueva planificación puede ser demasiado costoso o inviable. Por lo que sería deseable no tener que hacer de nuevo esta planificación desde cero o poder calcular los cambios solo para los módulos afectados. Pero esto puede acarrear problemas de compatibilidad entre módulos que han de tenerse en cuenta, por ejemplo, módulos que deben ser desplegados en el mismo proveedor, por lo que el cambio de proveedor de uno implica el cambio del otro.

Entonces, para gestionar los cambios necesarios a llevar a cabo durante la reconfiguración dinámica, sería interesante modelar a un alto nivel la variabilidad existente tanto en los requisitos de las aplicaciones *multi-cloud* como en los proveedores *cloud* y las dependencias entre ellos. Las Líneas de Producto Software *(SPL)* y los Modelos de Variabilidad posibilitan la gestión de heterogeneidad en los sistemas software. De hecho, hay varios trabajos recientes que aplican el paradigma de las *SPLs* a los entornos *cloud* [1, 2, 3, 4]. Por ejemplo, en [1] proponen un análisis basado en *SPLs* para permitir la migración de aplicaciones tradicionales al *cloud*. Para ello modelan la variabilidad de los proveedores con Modelos de Características (*Feature Models*). De un modo similar [2] definen los Modelos de Características Cloud (*Cloud Feature Models*) para usarlos en el proceso de selección de servicios *clouds*, que filtra y reduce el número de configuraciones de servicios válidas para cada aplicación. Por otro lado, en [3] proponen modelar las aplicaciones basadas en *cloud* como familias de *Sofware-as-a-Service* y definen un marco de trabajo sistemático y unificado par modelar los servicios *cloud* de un modo independiente al proveedor. En el caso de [4], para entornos *multi-cloud* usan el modelado de la variabilidad ortogonal para manejar la configuración de aplicaciones *multi-cloud*. Todos estos trabajos muestran los beneficios de aplicar el enfoque de *SPLs* mediante el uso de los modelos de variabilidad en tiempo de diseño y despliegue de las aplicaciones *cloud*. Pero ninguno aborda el uso de estos modelos en tiempo de ejecución para ayudar a la reconfiguración de estas aplicaciones.

Las Líneas de Producto Software Dinámicas (*DSLP*) producen software capaz de ser adaptado a cambios en tiempo real en el entorno. Las propuestas que usan este

enfoque para reconfigurar los sistemas software, suelen utilizan los modelos de variabilidad en tiempo de ejecución para obtener los cambios que han de llevarse a cabo durante la reconfiguración. Así, nosotros, al igual que hacen los trabajos previamente mencionados con el enfoque de SPL en tiempo de diseño y despliegue, proponemos usar el enfoque de las DSPL en tiempo de ejecución para manejar el proceso de reconfiguración de aplicaciones *multi-cloud*.

Entonces, en este trabajo de posicionamiento, además de usar los modelos de variabilidad para modelar los requisitos y las propiedades de las aplicaciones y las características de los proveedores, reflexionamos sobre la idoneidad de usarlos también en tiempo de ejecución. La idea es obtener automáticamente la nueva distribución de la aplicación en los proveedores para que siga funcionando correctamente en caso de producirse problemas o de violarse los requisitos. Las principales ventajas de usar los modelos de variabilidad para calcular la nueva distribución son dos. Por un lado, se pueden conservar fácilmente las especializaciones de esos modelos que contienen un subconjunto de configuraciones posibles (es decir sólo las configuraciones válidas que respeten los requisitos de las aplicaciones). De este modo, cuando sea necesaria una reconfiguración se puede buscar la nueva configuración entre ese subconjunto sin tener que hacerlo desde cero. Y por otro lado, como en los modelos de variabilidad es posible definir restricciones y dependencias entre los diferentes elementos, usaremos esta característica para definir las dependencias entre los módulos que deban ser desplegados en el mismo proveedor o en proveedores compatibles. Esto nos permite, que si el problema o el error afecta solo a un módulo, se puede obtener el nuevo proveedor para ese módulo de forma local, sin tener que calcular la distribución para la aplicación entera. Pero al tener en cuenta las dependencias, es fácil detectar si, al hacer este cambio local a dicho módulo, es necesario hacer también cambios en otro módulo.

Todos estos detalles los explicaremos con mayor profundidad en la siguiente sección, donde presentamos nuestra idea, que pretendemos seguir analizando y desarrollando. Además, ilustramos estas ideas iniciales con un ejemplo real para motivar la necesidad de abordar la reconfiguración dinámica de manera eficiente. Y por último, dado que estamos comenzando a discutir acerca de esta propuesta, varias cuestiones abiertas a ser discutidas serán presentadas.

2 Nuestro Enfoque

En la Figura 1, se muestra la visión general de nuestra propuesta. En esta se puede observar que se usan los modelos de variabilidad tanto para planificar la distribución de los módulos de una aplicación en los diversos proveedores como para obtener los cambios en esa distribución tras las sucesivas reconfiguraciones, aplicando el enfoque de *DSPLs*.

Estos modelos de variabilidad pueden ser integrados en la plataforma *SeaClouds*. Esta plataforma ya proporciona herramientas de descubrimiento de servicios *cloud*, despliegue y monitorización en tiempo real para comprobar si cumplen los requisitos de la aplicación y que podrían lanzar una reconfiguración si fuera necesario. En este punto es cuando se usarían en tiempo de ejecución los modelos de variabilidad para

obtener la nueva configuración, es decir, la nueva distribución de módulos de la aplicación en los diversos proveedores *cloud*.

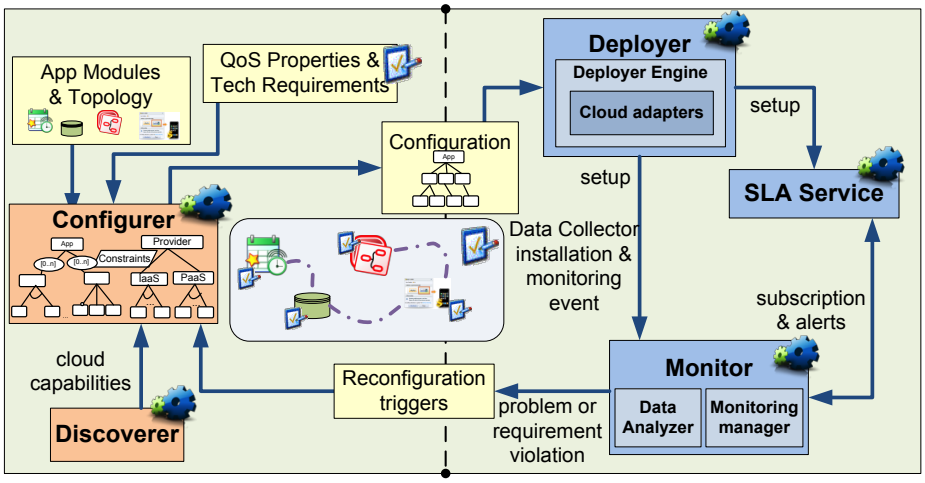

Figura 1. Proceso de Configuración y Reconfiguración de Aplicaciones Multi-cloud con Modelos de Variabilidad.

Para especificar los modelos de variabilidad vamos a usar un lenguaje de modelado de la variabilidad propuesto como estándar, *Common Variability Language (CVL)* [5]. Este lenguaje permite modelar la variabilidad (por ejemplo, mediante características opcionales o alternativas) de manera separada a un modelo base y tiene un conjunto de herramientas para manejar y analizar estos modelos. Dichas herramientas nos permitirán obtener automáticamente las configuraciones válidas que encajan con los requisitos de la aplicación. Para ello, modelamos por un lado la variabilidad de los proveedores, como se puede ver en la Figura 2, y por otro lado la variabilidad de las aplicaciones, como se ve en la Figura 3.

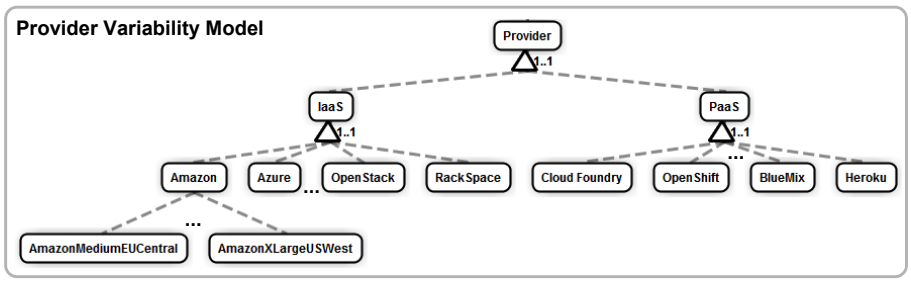

Figura 2. Modelo de Variabilidad de los Proveedores *Cloud*

La Figura 2 muestra el *Vspec tree* de CVL que representa el modelo de variabilidad de los proveedores (*Providers* en la figura), que pueden ser *IaaS* o *PaaS*.

Algunos ejemplos de *IaaS* que podemos ver en la figura son Amazon, Azure, OpenStack o RackSpace, mientras ejemplos de *PaaS* son Cloud Foundry, OpenShift, BlueMix o Heroku. Cada uno de los proveedores tiene unas características propias, definiendo su propia API, requisitos no funcionales, QoS, add-ons concretos, etc. Esto contempla información como su tecnología, peticiones por segundo, tiempo de respuesta o disponibilidad de la aplicación. Esta última información, se obtiene a través de *benchmarks*, pero pueden distar de la calidad ofrecida realmente y puede provocar una reconfiguración necesaria para adaptarse a los requisitos. La lista de proveedores soportada en el modelo de variabilidad habría que seguir completándola con todos los proveedores disponibles en el mercado, ya que aquí mostramos sólo algunos ejemplos. Además, los proveedores *cloud* están en constante evolución, por lo que tanto la lista de proveedores modelados como las características de cada uno pueden cambiar con el paso del tiempo. Esto es otra ventaja de los modelos de variabilidad, que son fácilmente evolucionables, añadiendo o modificando cada uno de los nodos de una forma sencilla y siempre modificando a su vez las restricciones o dependencias existentes. Esto lo trataremos en la siguiente sección como una cuestión abierta.

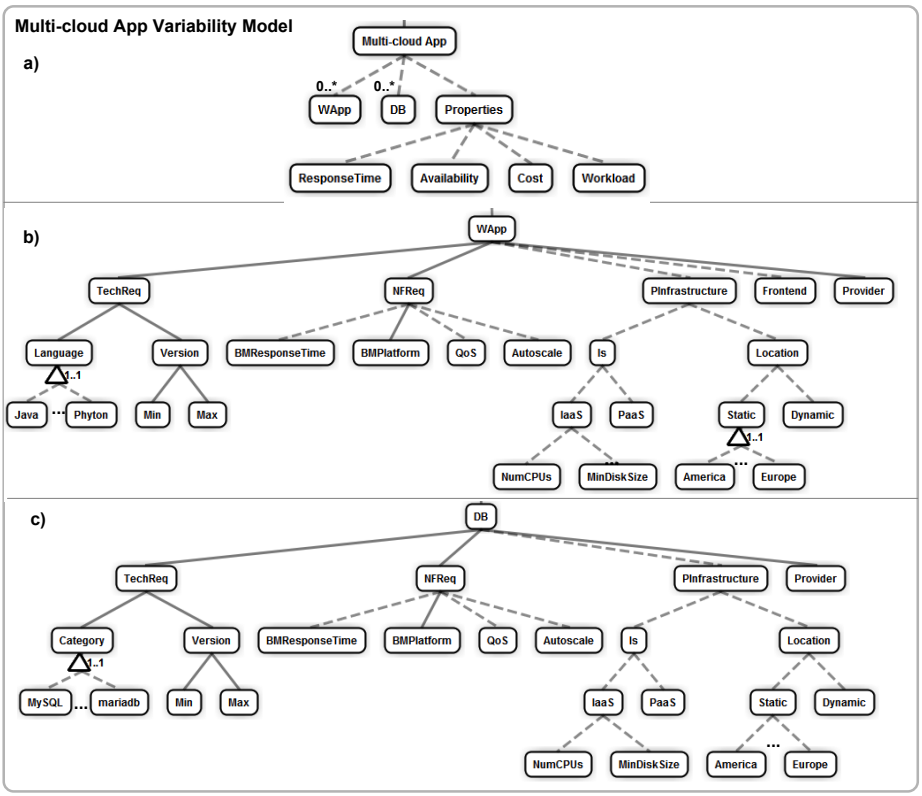

Figura 3. Modelo de Variabilidad de los Aplicaciones *Multi-cloud*

En la Figura 3, podemos ver el *Vspec tree* de CVL para las aplicaciones *multi-cloud*. En el caso de *SeaClouds*, se considera que una aplicación puede estar compuesta de uno o más módulos, pudiendo ser aplicaciones web (*WApp*) o módulos de bases de datos (*DB*). Esto se representa con dos características o con cardinalidad (0..*) en la Figura 3.a. Además, en este modelo de variabilidad se especifican (mediante características con atributos) las propiedades que esta aplicación requiere de modo general (p.ej., tiempo de respuesta). Luego, para cada uno de los módulos se especifican sus propiedades y requisitos. Por ejemplo, en el caso de las bases de datos, podemos considerar la categoría de la base de datos (MySQL, MongoDB, MariaDB, PostgreSQL, etc., modelado como características alternativas XOR en el modelo de variabilidad de la Figura 3.c), o el número de CPUs o espacio en disco de la máquina, en el caso de *IaaS*, donde también se puede elegir si se desea localización estática (Europa, América, …) o dinámica. Para las aplicaciones web, se consideran valores como si son *frontend* o no (modelado como característica opcional en el modelo de variabilidad de la Figura 3.b), el lenguaje o tecnología (Java, Python, Ruby, Net, PHP), o bien si se hace *autoscaling* del módulo o no (que también se puede considerar en bases de datos). Además, a cada módulo se le ha de asignar un proveedor, por lo que el *Vspec tree Provider* de la Figura 2 se referencia también en el modelo de variabilidad de cada módulo, como se puede ver en la Figura 3.b y 3.c.

De este modo, usando estos árboles especificados en CVL y las restricciones que definimos (usando las *constraints* de CVL) entre los requisitos de la aplicación y de cada módulo, y las características propias de los proveedores, podemos obtener las configuraciones que cumplan estas restricciones (es decir, la distribución de módulos en los proveedores adecuados). Un ejemplo de estas restricciones sería que si para un módulo se selecciona la localización estática para un *IaaS* y se requiere que sea en Europa, esto implica que entonces no se puede seleccionar *AmazonXLargeUSWest* para ese módulo. Al conjunto de configuraciones válidas se le llama especialización en terminología de modelos de características. En esta especialización se habrán podado muchas ramas del árbol de CVL, ya que sólo contiene las configuraciones que son posibles para las restricciones de la aplicación y sus módulos. Así, esta especialización será la que se use para, en caso de ser necesaria una reconfiguración dinámica, obtener la siguiente configuración válida, evitando el cálculo desde cero.

Para ilustrar nuestro enfoque, usamos un caso real de uno de los casos de uso empleados en el proyecto *SeaClouds*, en concreto el caso de uso de Atos, Softcare [6]. Por ejemplo, en la Figura 4 se muestra una especialización para esta aplicación que se compone de cinco módulos, tres aplicaciones web y dos bases de datos. También vemos en la Figura 4.a las propiedades generales requeridas para la aplicación (p.ej. tiempo de respuesta de 2000ms). En la Figura 4.b se muestra la especialización para uno de los módulos de aplicación web, que es una aplicación *frontend*, que usa Java y requiere que el provider sea *PaaS* además de especificar otros requisitos. Usando las restricciones previamente definidas, tras analizar los requisitos globables de la aplicación y los requisitos locales de este módulo, obtendremos la especialización de la Figura 4.b, donde aún quedan abiertas la selección de algunas de las ramas del árbol (por ejemplo decidir entre desplegar ese módulo en Cloud Foundry o desplegarlo en BlueMix), pero muchas otras ramas (por ejemplo, todos los proveedores IaaS y los demás proveedores PaasS, como Heroku y OpenShift) se habrán podado.

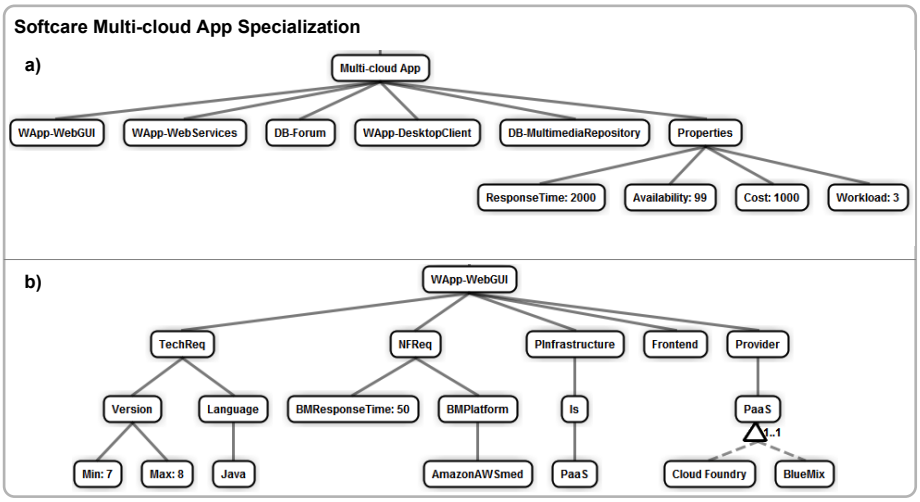

Figura 4. Especialización del Modelo de Variabilidad de una Aplicación *Multi-cloud*

Para este módulo, en esta especialización hay sólo dos configuraciones válidas, una donde se despliega en Cloud Foundry y otra donde se despliega en BlueMix. Imaginemos que se está ejecutando la configuración donde se ha desplegado en Cloud Foundry, pero dicho proveedor debido a un error o a una congestión no proporciona el tiempo de respuesta requerido por la aplicación. Esta violación se puede detectar gracias la funcionalidad de monitorización en conexión con el componente que gestiona los acuerdos con los providers (*Service Level Agreements, SLA*) de *SeaClouds*. Entonces, de entre las posibilidades que teníamos en la especialización de la Figura 4.b, utilizando CVL se escoge otra configuración válida para ese módulo que no contenga al proveedor afectado. En este caso la única configuración válida sería la del proveedor BlueMix, por lo que la diferencia entre estas dos configuraciones implica sólo migrar el módulo WApp-WebGUI con su código correspondiente de Cloud Foundry a BlueMix. Esto se realiza usando la funcionalidad de migración de un módulo soportada por *SeaClouds*, sin necesidad de parar el resto de la aplicación.

Ahora imaginemos que el proveedor del módulo de base de datos del repositorio multimedia (Figura 4.a) es el que no cumple las restricciones, entonces sólo habría que chequear qué otro proveedor es adecuado para sus requisitos y los de la aplicación. Pero revisando sus dependencias, encontramos que se solicitó que los dos módulos de bases de datos estuvieran alojados en el mismo proveedor. Entonces el módulo DB-Forum debe estar en el mismo proveedor que el repositorio multimedia DB-MultimediaRepository. Así que hay que considerar esta dependencia durante el proceso de migración del repositorio, migrando también el módulo correspondiente al foro y su código. Al tener especificadas las restricciones entre módulos y proveedores en el mismo modelo de variabilidad, estas dependencias se detectan fácilmente, pudiendo actuar en consecuencia. Pero así lo interesante de nuestra propuesta, es que la reconfiguración se puede hacer de forma local al módulo con problemas sin tener

que computar de nuevo todas las posibilidades, a diferencia de lo que actualmente se debe hacer con otras plataformas que dan la posibilidad de migrar módulos, como la funcionalidad proporcionada por *SeaClouds*.

3 Cuestiones Abiertas

Tras las ideas expuestas en este trabajo de posicionamiento y reflexión, donde comenzamos a explorar el uso del enfoque de *DSPL* para abordar la reconfiguración dinámica de aplicaciones *multi-cloud*, queda aún mucho trabajo futuro y cuestiones abiertas.

En primer lugar, como hemos comentado en la sección anterior, lo primero sería completar todos los modelos de variabilidad, especialmente contemplando todos los proveedores del mercado (o al menos una lista importante de ellos soportados por soluciones actuales en esta línea) y sus características.

En segundo lugar, sería deseable automatizar por completo el proceso de configuración y reconfiguración, para lo que deberíamos poder trasladar la información de una configuración válida elegida que está modelada en CVL a un modelo de especificación estándar, como el que usa *SeaClouds*, TOSCA [7]. De esta manera, se conectaría directamente con los mecanismos ya proporcionados por *SeaClouds* para llevar a cabo el despliegue en entornos *multi-cloud*. Creemos que hacer el mapeado CVL2TOSCA es una tarea que se puede abordar, pero se nos quedan cuestiones pendientes como si también mapearíamos las reglas de monitorización y los acuerdo a nivel de servicio que actualmente *SeaClouds* los incluye en TOSCA. Una ventaja de tener este mapeado de CVL a TOSCA es que para la reconfiguración, detectando las diferencias [8] entre la configuración que estaba ejecutándose y la que se va a ejecutar, se puede obtener automáticamente qué partes del modelo TOSCA hay que cambiar.

Por otro lado, en este trabajo sólo hemos mencionado que puede ser necesaria una reconfiguración dinámica cuando los proveedores no funcionan bien o no cumplen una serie de requisitos, pero hay más aspectos que pueden cambiar en tiempo de ejecución. Por ejemplo, sería deseable dar la posibilidad a los usuarios de cambiar los requisitos, como el coste de un servicio, o la disponibilidad o tiempo de respuesta de un proveedor *cloud*, y esto también lanzaría una reconfiguración para adaptar la aplicación a los nuevos requisitos. O por otro lado, como ya avanzamos en la sección anterior, los proveedores también evolucionan con el tiempo y sus especificaciones (por ejemplo, la versión de Java con la que es compatible) pueden cambiar, lo que también podría conllevar una reconfiguración (por ejemplo, si la versión de Java del módulo no está ahora en el rango de versiones de Java soportadas por el proveedor en el que se estaba ejecutando, por alguna actualización en dicho proveedor). Por último, también pueden aparecer nuevos proveedores con características muy ventajosas (por ejemplo, con bajo coste y mínimo tiempo de respuesta) y sería deseable ofrecer al usuario migrar los módulos que fueran posibles a dicho proveedor. Todo esto no está cubierto por plataformas del tipo de *SeaClouds*, pero usando modelos de variabilidad es posible abordar estos aspectos, con la especificación de nuevas configuraciones en tiempo de ejecución, pudiendo controlarse las diferencias concretas de manera sencilla. Soluciones en las que se presentan las ventajas de estas técnicas se pueden

ver en trabajos previos sobre evolución de modelos de variabilidad de dos de los autores de este trabajo [8].

En resumen, como se ha mencionado en este trabajo inicial, el enfoque aquí planteado pretende abordar un problema en un campo en el que quedan muchas cuestiones abiertas que pueden ayudar en el tema de los entornos *multi-cloud*, que está muy de auge actualmente y en el que se necesitan muchos esfuerzos.

Referencias

1. García-Galán J., Rana O., Trinidad P. and Ruiz-Cortés A. (2013). Migrating to the Cloud - A Software Product Line based Analysis.In Proceedings of the 3rd International Conference on Cloud Computing and Services ScienceISBN 978-989-8565-52-5, pages 416-426
2. Erik Wittern, Jörn Kuhlenkamp, and Michael Menzel. 2012. Cloud service selection based on variability modeling. In Proceedings of the 10th international conference on Service-Oriented Computing (ICSOC'12), Chengfei Liu, Heiko Ludwig, Farouk Toumani, and Qi Yu (Eds.). Springer-Verlag, Berlin, Heidelberg, 127-141
3. M. A. Matar, R. Mizouni and S. Alzahmi, "Towards Software Product Lines Based Cloud Architectures," Cloud Engineering (IC2E), 2014 IEEE International Conference on, Boston, MA, 2014, pp. 117-126.
4. Jamshidi, Pooyan and Pahl, Claus: Orthogonal Variability Modeling to Support Multi-cloud Application Configuration. In Advances in Service-Oriented and Cloud Computing: Workshops of ESOCC 2014, Manchester, UK, September 2-4, 2014.
5. Haugen, Ø. et al.: Adding Standardized Variability to Domain Specific Languages. In Proc. of 12th Software Product Line Conference. IEEE Computer Society, (2008).
6. SeaClouds Project. Deliverable D6.3.3 Case Studies Final implementation (SeaClouds Consortium), April 2016.
7. OASIS. TOSCA 1.0 (Topology and Orchestration Specification for Cloud Applications), Version 1.0, http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.pdf, 2013.
8. Gámez, N., Fuentes, L.: Architectural Evolution of FamiWare using Cardinality-Based Feature Models. Journal of Information and Software Technology 55(3): 563-580, (2013)