Retos Actuales en el Desarrollo de Lenguajes Específicos del Dominio

Javier Cánovas, Salvador Trujillo

Retos Actuales en el Desarrollo de Lenguajes Específicos del Dominio

Javier Luis Cánovas Izquierdo, Salvador Trujillo

AtlanMod, École des Mines de Nantes – INRIA – LINA, Nantes, Francia
javier.canovas@inria.fr
IK4-IKERLAN, Mondragón, España
strujillo@ikerlan.es

Resumen Los Lenguajes Específicos del Dominio (*Domain-Specific Languages*, DSLs) son lenguajes especialmente concebidos y diseñados para trabajar con los conceptos de un dominio particular. La potencia ofrecida por los DSL para definir soluciones en un dominio fomenta la productividad y mantenibilidad, entre otros beneficios. En los últimos años, los avances de las herramientas de desarrollo de DSLs han permitido que la creación de nuevos lenguajes sea una alternativa real en determinados dominios dentro del desarrollo de software industrial. Sin embargo, creemos que todavía existen limitaciones que dificultan una adopción más generalizada por parte de la industria. En este trabajo presentamos nuestra experiencia en el desarrollo de varios DSLs en un contexto real industrial, como son los sistemas embebidos, donde ponemos en relieve las dificultades encontradas. A partir de estas limitaciones identificamos los principales retos que creemos podrían mejorar la forma de crear DSLs, favoreciendo su adopción industrial.

1. Introducción

Los Lenguajes Específicos del Dominio (*Domain-Specific Languages*, DSLs) son lenguajes especialmente creados para realizar tareas en un dominio determinado. El uso de DSLs facilita la creación de soluciones en un dominio particular, favoreciendo la productividad y el mantenimiento, entre otros beneficios [1].

En los últimos años, el avance en las herramientas de desarrollo de DSLs ha permitido que la creación de nuevos lenguajes sea una alternativa real para abordar problemas en un dominio específico. De esta forma, alejándose de un contexto puramente académico, la decisión de crear y utilizar DSLs en ámbitos industriales parece ser una opción a valorar cada vez más frecuente. Aunque el número de trabajos que presentan aplicaciones industriales de DSLs es creciente [2–5], creemos que todavía existen limitaciones a superar para conseguir una verdadera adopción de esta tecnología por la industria. Así, la inexistencia de un proceso estándar de definición de DSLs o el limitado soporte a la evolución continua del lenguaje, sus herramientas y artefactos asociados son aspectos fundamentales que amenazan las promesas de productividad y mantenimiento.

En este sentido, en este trabajo presentamos nuestro punto de vista acerca de los principales retos a superar para facilitar la adopción de la ingeniería de DSLs por la industria. Este artículo es fruto de la colaboración de un grupo de investigación y un centro de investigación tecnológica con proyectos industriales, los cuales aportan una visión teórica/académica y práctica en la creación de DSLs, respectivamente. En concreto, en

este artículo analizamos los principales retos encontrados en el desarrollo de DSLs en el dominio de los sistema embebidos, en particular, en los sectores de la energía, transporte y salud. Aunque las conclusiones descritas no son totalmente generalizables, creemos que pueden servir para identificar posibles retos futuros en la ingeniería de DSLs.

Este artículo está organizado de la siguiente manera. La Sección 2 describe cómo se desarrollan actualmente los DSLs mientras que la Sección 3 muestra las principales limitaciones que hemos encontrado cuando se lleva el proceso de desarrollo a la práctica así como los retos identificados. Finalmente la Sección 4 presenta las conclusiones y el trabajo futuro.

2. Situación Actual en el Desarrollo de Lenguajes Específicos del Dominio

Un DSL está compuesto por tres elementos [6]: sintaxis abstracta, sintaxis concreta y semántica. La sintaxis abstracta define los conceptos principales del lenguaje, sus relaciones y un conjunto de restricciones; típicamente se implementa en la forma de un metamodelo. La sintaxis concreta define la notación del lenguaje, que puede ser textual y/o gráfica. Finalmente, la semántica del lenguaje se define generalmente por medio de una aproximación traslacional aunque en la práctica se realiza informalmente. Además, un DSL se proporciona con un conjunto de utilidades (p. ej., validadores, editores con auto-completado, resaltado de sintaxis, etc.) que son cruciales para su éxito.

En la actualidad, no existe una forma estándar de crear DSLs, solamente recomendaciones acerca de las fases de desarrollo existentes y qué realizar en cada una de ellas (p. ej., [1, 5–7] entre otros). En este sentido, [1] es quizás el trabajo más representativo, donde se identifica un proceso de desarrollo compuesto de cinco fases, que son: decision, análisis, diseño, implementación y despliegue. La necesidad de crear un nuevo lenguaje se realiza en la fase de decisión mientras que en la fase de análisis se realiza el estudio del dominio, donde se identifican los principales conceptos y relaciones. A continuación, el proceso de creación del nuevo lenguaje se realiza en las fases de diseño e implementación, donde se definen los componentes principales del DSL (i.e., sintaxis abstracta y concreta, y semántica) así como las herramientas asociadas (p. ej., editores y/o generadores). Finalmente el lenguaje se entrega al cliente en la fase de despliegue.

3. Limitaciones y Retos Encontrados

Aunque el proceso anterior sirve de guía para crear DSLs, adolece de ser demasiado teórico y no considera aspectos fundamentales que aparecen en la práctica, como el papel que juega el cliente, la evolución del DSL o la calidad de sus utilidades. Estos aspectos salen a relucir cuando se aplica este proceso en un contexto real e industrial, como fue nuestra experiencia, la cual presentamos en esta sección.

IK4-IKERLAN es un centro de investigación dedicado a proyectos de investigación aplicada y transferencia industrial, desarrollando una labor significativa en el campo de los sistemas embebidos para diversos sectores: transporte, energía, salud, industrial, etc. Durante los últimos años, el centro ha trabajado en un conjunto de proyectos donde se han utilizado DSLs para facilitar el desarrollo de sistemas de control industriales.

En este trabajo mostramos la experiencia adquirida en el desarrollo de DSLs para abstraer el sistema de control en sistemas embedidos de los sectores de transporte, energía y salud. En particular, se crearon tres DSLs para: describir los sistemas de control de tracción en los ferrocarriles, especificar los componentes en aerogeneradores y modelar aparatos médicos. Estos DSL permiten generar código adaptado a la arquitectura embebida objetivo, favoreciendo la productividad en los procesos de desarrollo. Todos los DSL fueron desarrollados en la platforma Eclipse haciendo uso del *framework* de modelado EMF[1]. A continuación mostramos la aplicación del proceso de desarrollo en la práctica y finalmente los retos de investigación identificados.

3.1. El Proceso en la Práctica

En esta sección presentamos los aspectos más destacables que surgieron durante el desarrollo de los lenguajes, enmarcando cada uno de ellos en la fase del proceso adecuada. Durante el desarrollo de estos lenguajes la empresa aplicó un proceso parecido al descrito en la Sección 2.

Fase de decisión.

Proceso software del cliente. De forma parecida a como se recomienda en trabajos como [5, 8], la fase de decisión estuvo fuertemente influenciada por el estudio del proceso de desarrollo software del cliente. La principal estrategia que se siguió fue identificar tareas repetitivas o que siguen un patrón determinado para automatizarlas por medio de un DSL. Por ejemplo, en el DSL para ferrocarriles, se identificó una línea de producto en torno a una arquitectura común con puntos de variación (p. ej., evaluadores de estado o subsistemas de control).

Competencias del cliente. Un aspecto importante fue medir el esfuerzo de formación necesario para que el DSL sea utilizado por la empresa cliente. En nuestro caso, dado que el usuario final tenía un perfil técnico se consideró viable desarrollar DSLs integrados en Eclipse y que utilizaran sus vistas (p. ej., editores de árbol, *outline*, etc.).

Fase de análisis.

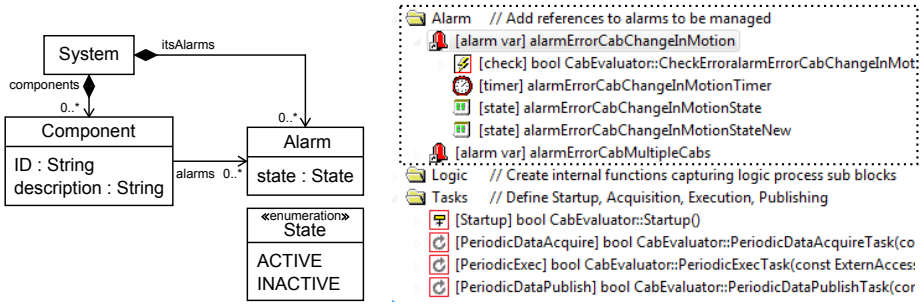
Evolución continua del dominio. La parte relativa al análisis fue quizás la que más atención recibió, ya que en la práctica, es muy difícil identificar el dominio de aplicación en una sola iteración, como nos occurrió durante el desarrollo de los tres DSLs. Esto fue debido fundamentalmente a la baja disponibilidad del cliente y a que no era consciente de la especificidad de su dominio, dificultando la identificación de sus límites.

Fase de diseño.

Adecuar la sintaxis. En esta fase un aspecto fundamental fue identificar la sintaxis concreta más adecuada para el cliente. En los casos que nos ocuparon, una representación arbórea fue suficiente dado que el cliente tenía conocimientos técnicos. En el caso del DSL para aerogeneradores también se definió una sintaxis gráfica con el objetivo de facilitar la identificación de los componentes del sistema.

Diseñar para facilitar la evolución. El aspecto quizás más importante a tratar, y relacionado con la fase anterior, fue cómo hacer evolucionar el lenguaje cuando el dominio cambia. Añadir nuevos conceptos o relaciones implicaba cambiar la definición

[1] http://eclipse.org/emf



(a) (b)

Figura 1. Parte del (a) metamodelo y (b) ejemplo de modelo de la sintaxis abstracta del DSL para control de trenes que permite la definición de alarmas.

del lenguaje y, consecuentemente, las herramientas dependientes, siendo un esfuerzo considerable. En nuestro caso particular, se optó por intentar mantener el metamodelo de sintaxis abstracta a muy alto nivel, de forma que nuevos conceptos del dominio serían representados como instancias de dicho metamodelo. Por ejemplo, en el DSL para sistemas de control de trenes existen diferentes tipos de alarmas (p. ej., control de temperatura, control eléctrico, etc.). En vez de representar la jerarquía de alarmas en el metamodelo de sintaxis abstracta, se optó por definir solamente el concepto de alarma. De esta forma, los tipos de alarma se representan como instancias de la sintaxis abstracta. La Figura 1 muestra una parte del metamodelo de sintaxis abstracta mientras que la Figura 1b muestra un ejemplo del DSL que define diferentes alarmas.

Fase de implementación.

Evolución de las herramientas. Aunque la decisión de diseño tomada en la fase anterior facilitaba hacer evolucionar el dominio fácilmente, todavía se hacía necesario adaptar la infraestructura y herramientas asociadas (fundamentalmente transformaciones modelo-a-código en nuestro caso), lo cual requería un esfuerzo importante.

Adaptación de la herramienta al cliente. Dado que las herramientas de desarrollo de DSL generan parte de la infraestructura del lenguaje (EMF genera el API Java así como el editor arbóreo para el DSL), durante la fase de implementación se modificó dicha infraestructura para mejorar su usabilidad o apariencia (p. ej., creación de asistentes, menús contextuales, etc.). Esta adaptación también tuvo en cuenta la integración de la herramienta con la arquitectura software del cliente (p. ej., en el DSL para ferrocarriles, enlazando con el sistema de comunicaciones o la interacción de buses de campo).

Fase de despliegue.

Validación de la arquitectura. La fase de despliegue se dedicó a la validación y prueba de los DSLs en los sistemas. En el caso particular de los DSLs desarrollados, donde los DSLs se encargaban de generar el código para diferentes arquitecturas embebidas, el problema principal fue como validar automáticamente cada una de las posibles combinaciones en las pruebas de código.

Documentación. Un aspecto fundamental en la entrega del DSL al cliente es la documentación asociada. Para los tres DSLs desarrollados, la principal documentación fue un conjunto de ejemplos y guías de uso de los generadores de código de cada DSL.

3.2. Retos

A partir de los aspectos descritos en la sección anterior identificamos los principales retos de investigación en el desarrollo de DSLs. Los elementos se listan de mayor a menor prioridad de acuerdo a una encuesta realizada en la empresa (en paréntesis se muestra la prioridad dada por el grupo de investigación)[2]:

1°. Involucrar al usuario final (1°). La participación del usuario final en el proceso de desarrollo permitiría una identificación del dominio más eficaz y facilitaría la adaptación del lenguaje a sus necesidades al permitirle tomar parte en las decisiones de diseño. De forma parecida a las propuestas ágiles[3], una posible solución es involucrar activamente al cliente en cada fase del desarrollo del lenguaje con la intención de validar y reducir el tiempo de desarrollo. En este sentido, trabajos como [9] ofrecen una aproximación orientada a favorerer la colaboración entre desarrolladores y usuarios finales en la creación de lenguajes.

2°. Soporte a la evolución continua del dominio (2°). Las herramientas de desarrollo de DSLs actuales permiten generar parte de la infraestructura del DSL pero ofrecen un soporte pobre a la evolución del lenguaje. Por ejemplo, normalmente el proceso de re-generación no tiene en cuenta cambios manuales (p. ej., GMF sobreescribe los modelos en cada generación) o herramientas asociadas (p.ej., generadores que dependen del DSL), dando lugar a los problemas detectados fundamentalmente en las fases de análisis y diseño. Así, resulta necesario potenciar mecanismos que permitan identificar qué elementos modificar cuando el dominio cambia (p.ej., qué elementos del editor se ven afectados al incluir un nuevo concepto). En este sentido, trabajos como [10] ofrecen soporte a la evolución, aunque únicamente de las transformaciones de modelos.

3°. Validación del código generado por el DSL (5°). Aunque existen trabajos (p. ej., [5]) que presentan algunas técnicas para validar DSLs, un aspecto todavía no tratato en profundidad es cómo automatizar la validación del código generado por un DSL, lo que facilitaría la fase de despliegue. En particular, es necesario estudiar métodos para validar las posibles combinaciones de código que pueden generarse, por ejemplo, identificando cuáles son válidas para el sistema destino.

4°. Integración con el sistema del cliente (3°). Normalmente la integración con otras aplicaciones se realiza por medio de interfaces externas, tal y como detectamos en la fase de implementación. En este sentido, propuestas como [11,12] podrían adaptarse para ofrecer mecanismos que definan las interacciones entre los DSLs y dichas interfaces.

5°. Necesidad de un marco de referencia para el proceso (6°). Tal y como se ha comentado en la sección 2, existen varias propuestas para guiar el desarrollo de DSLs pero ninguna de ellas estándar. Por este motivo creemos que la definición de un marco de referencia que presente las alternaticas disponibles durante el desarrollo de DSL y recomendaciones acerca de cómo seleccionarlas (p. ej., la forma de definir la semántica o cuándo elegir una sintaxis textual o gráfica) mejoraría su adopción por la industria.

---

[2] En la encuesta participaron nueve desarrolladores de IK4-IKERLAN y nueve miembros de AtlanMod. Cada grupo elaboró una lista priorizada con los retos identificados.

[3] Agile Manifesto – http://agilemanifesto.org

6°. Documentación del lenguaje (4°). Mientras que en el desarrollo de sistemas software existe cierto consenso a la hora de documentar el sistema, no está claro en el caso de los DSLs, como nos ocurrió en la fase de despliegue. Para abordar este reto, creemos que sería necesario un mayor número de experiencias de uso y creación de DSLs en la industria, permitiendo identificar la forma más adecuada de documentar un lenguaje.

4. Conclusiones y Trabajo Futuro

Este artículo presenta los retos encontrados durante el desarrollo de tres DSLs en el ámbito de sistemas embebidos. Aunque las limitaciones descritas surgen de un dominio determinado, creemos que sirven para identificar posibles líneas de trabajo futuras que faciliten la adopción de los DSLs por la industria.

Actualmente estamos trabajando en el estudio del soporte a la evolución continua del dominio y la participación de los usuarios finales en el proceso. En particular, estamos trabajando en cómo mantener sincronizados los elementos del DSL con su infraestructura (p. ej., evolución automática de los editores cuando el DSL cambia), así como en la participación de los usuarios finales evitando las tareas más técnicas (p. ej., utilizando ejemplos del lenguaje en vez de su sintaxis abstracta).

Referencias

1. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv. **37** (2005) 316–344
2. Clark, T., Muller, P.A.: Exploiting model driven technology: a tale of two startups. Software & Systems Modeling **11**(4) (2012) 481–493
3. Selic, B.: What will it take? A view on adoption of model-based methods in practice. Software & Systems Modeling **11**(4) (2012) 513–526
4. Aranda, J., Damian, D., Borici, A.: Transition to Model-Driven Engineering What Is Revolutionary, What Remains the Same? In: MODELS conf. Volume 7590., LNCS (2012) 692–708
5. Voelter, M.: DSL Engineering. dslbook.org (2013)
6. Kleppe, A.: Software Language Engineering: Creating Domain-Specific Languages Using Metamodels. Addison Wesley (2008)
7. Ceh, I., Crepinsek, M., Kosar, T., Mernik, M.: Ontology driven development of domain-specific languages. Computer Science and Information Systems **8**(2) (2011) 317–342
8. Kelly, S., Pohjonen, R.: Worst practices for domain-specific modeling. IEEE Software **26**(4) (2009) 22 –29
9. Cánovas Izquierdo, J.L., Cabot, J.: Enabling the collaborative definition of DSMLs. In: CAiSE conf., pendiente de publicación (2013)
10. García, J., Díaz, O.: Model Transformation Co-evolution: A Semi-automatic Approach. In: SLE conf. Volume 7745., LNCS (2012) 144–163
11. Cánovas Izquierdo, J.L., Jouault, F., Cabot, J., García Molina, J.: API2MoL: Automating the building of bridges between APIs and Model-Driven Engineering. Information and Software Technology **54** (2012) 257–273
12. Sánchez Cuadrado, J., Guerra, E., Lara, J.D.: The Program is the Model: Enabling Transformations@run.time. In: SLE conf. Volume 7745., LNCS (2012) 1–20