Una exploración sobre cómo DSL existentes pueden beneficiarse de GenAI

Antonio Pérez-Serrano, Jesús García Molina[1], José Ramón Hoyos Barceló[1], and María José Ortín Ibáñez[1]

Universidad de Murcia, Murcia, España
{a.perezserrano1,jmolina,jose.hoyos,mjortin}@um.es

Resumen Al igual que otras áreas de la ingeniería del software, los lenguajes específicos de dominio (DSL) también se ven afectados por la aparición de los modelos de lenguaje grandes (LLM) y la inteligencia artificial generativa (GenAI). Actualmente, es posible entrenar modelos generales para que comprendan tanto la sintaxis como la semántica de cualquier DSL. De este modo, se facilita la realización de operaciones sobre DSL. Este trabajo explora esta capacidad de los LLM a través de una familia de DSL genéricos destinados a definir esquemas de bases de datos (Athena) y operaciones de evolución de esquemas (Orion). El objetivo es mostrar cómo los LLM han sido aprovechados para implementar operaciones de conversión de lenguajes que involucran a Athena y Orion. Finalmente, se extraen conclusiones sobre los beneficios de seguir usando DSL en la era de los LLM.

Keywords: DSL · LLM · Aprendizaje DSL · Orion · Athena

1. Motivación y Objetivo

A lo largo de la historia de la Ingeniería del Software, los Lenguajes Específicos del Dominio (DSL) han proporcionado una abstracción superior a los lenguajes de propósito general. Sin embargo, con la llegada de los modelos de lenguaje (LLM) y la IA Generativa (GenAI) surge la posibilidad de generar automáticamente artefactos de software desde lenguaje natural, lo que podría desplazar la función de los DSL. Dado que crear un DSL es costoso, algunos plantean la idea de reemplazarlos con prompts en lenguaje natural. No obstante, esta sustitución no resulta tan simple, pues los DSL ofrecen un control y una precisión que no siempre se logran únicamente con prompts y texto libre.

La relación entre DSL y LLM/GenAI ha sido abordada en algunos trabajos. Por ejemplo, B. Böckelund [1] ha notado la importancia de considerar la distancia entre el nivel de abstracción del prompt y el lenguaje destino: a menor distancia, más fácil será escribir el prompt que será más corto y simple. L. Netz et al. [7] han presentado un enfoque para entrenar un LLM genérico, en concreto GPT-4, con el objetivo de generar scripts de un DSL a partir de lenguaje natural, de modo que usuarios no expertos pueden crear aplicaciones Web. Una herramienta genérica DSL-Xpert capaz de generar scripts de cualquier

DSL a partir de lenguaje natural ha sido desarrollada por D. García-González et al. [4]. Para ello, un LLM se entrena con la gramática (*grammar prompting*) y con ejemplos (*few-shot learning*). Estos tres trabajos ilustran un interés por la generación de modelos o scripts de DSL utilizando LLM. Por otro lado, M. Mosthaf et al. [6] han presentado un prototipo de asistente basado en GPT-4o para facilitar la creación de un DSL: su gramática y ejemplos. En la literatura, hasta donde conocemos, no existe un survey que analice cómo los DSL y LLM pueden beneficiarse mutuamente y cuál es el futuro del manejo de modelos con los LLM. En este artículo corto, presentamos un trabajo de experimentación que estamos realizando para abordar la elaboración de ese survey, lo cual sería un paso previo a la construcción de un framework de manejo de modelos en la tesis doctoral del primer autor. En la siguiente sección describiremos los experimentos realizados con la familia de DSL creados alrededor del modelo de datos unificado U-Schema en el grupo ModelUM [3]. U-Schema es un metamodelo destinado a representar esquemas de bases de datos relacionales y NoSQL. Su finalidad es facilitar el diseño e implementación de herramientas genéricas que soporten tareas de bases de datos de manera independiente al modelo de datos. En ese contexto se han definido varios DSLs y aquí trataremos con dos de ellos: Athena [5] y Orion [2]. Ambos son lenguajes genéricos para expresar esquemas y operaciones de evolución de esquemas, respectivamente. Finalmente, en la tercera sección extraeremos algunas conclusiones.

2. Experimentos con Athena y Orion

Se ha utilizado GPT-4o como LLM y se ha usado la misma estrategia de entrenamiento para los dos DSL: *aprendizaje basado en prompts*. El entrenamiento ha estado dirigido a disponer de dos modelos de lenguajes, uno para Orion y otro para Athena, capaces de: (i) *Conversión bidireccional de esquemas Athena* a esquemas SQL, MongoDB y Cassandra, con validaciones sintácticas y semánticas. (ii) *Conversión bidireccional de script Orion* a operaciones de cambio de esquema en SQL, CQL y MongoDB, y (iii) *Generación de esquemas Athena a partir de especificaciones en lenguaje natural*. Las dos primeras operaciones formaban parte de una lista de tareas pendientes que pensábamos implementar por medio de transformaciones de modelos. Todo el material utilizado en el entrenamiento y validación está disponible en un repositorio de GitHub [1]. La estrategia de aprendizaje consistió en los siguientes 4 pasos que se realizaron con ChatGPT:

1. *Prompt del Sistema*: Primero se describió al modelo tanto el problema como la estrategia para resolverlo.
2. *Grammar Prompting*: Se instruyó al modelo con las reglas sintácticas y reglas para obtener modelos bien formados de Athena y Orion, las cuales fueron extraídas de su definición en Xtext.
3. *Uso de Documentación Técnica*: Se utilizaron artículos científicos [5,2] que incluían el metamodelo y describían los DSL con ejemplos, con el fin de reforzar la comprensión del modelo.

[1] `https://github.com/modelum/jisbd2025-DSLs`

4. *Few-Shot Learning*: Se presentaron 5 ejemplos extensos de conversiones entre Athena y los lenguajes SQL, CQL para Cassandra y Mongoose para MongoDB, en los dos sentidos. Eso supone un total de 30 ejemplos. Cabe indicar que Athena es más complicado que Orion, dado que la dificultad de especificar operaciones de cambio de esquema es menor que la de especificar esquemas. Por ello, en el caso de Orion, se utilizaron sólo dos ejemplos con operaciones Orion de todos los tipos y para generar código SQL, CQL, MongoDB y Neo4J. En este caso, el número total de ejemplos fue de 8.

Durante el entrenamiento de cada DSL, el modelo no sólo entendió la sintaxis, sino que planteó dudas sobre aspectos como la gestión de versiones en Athena o la combinación de rangos con expresiones regulares. Ante nuestras aclaraciones, sugirió la necesidad de abordar dichas cuestiones. Los ejemplos incluyeron la mayoría de elementos de Athena y Orion: agregados, referencias, jerarquías y relaciones M:N. Al abarcar menos casos en Orion, las pruebas requirieron más iteraciones para validar la aplicación de sus operaciones de evolución.

Para validar el aprendizaje, primero se lanzaron un conjunto de casos de prueba a los dos modelos entrenados. Los modelos demostraron la capacidad de responder correctamente a las pruebas para las 3 operaciones indicadas al inicio de esta sección. Para validar la corrección de los scripts Athena y Orion generados, se utilizó una herramienta de validación ya disponible para estos DSLs, la cual comprueba que se satisface la gramática y las reglas de validación tipo OCL declaradas. Cuando el resultado eran scripts SQL, CQL y MongoDB se probaron sobre las correspondientes bases de datos.

3. Conclusiones

A partir de nuestra experiencia, hemos identificado varias formas en las que el uso de DSLs puede beneficiarse de los LLM y viceversa.

Facilitar operaciones con modelos o scripts de un DSL. Una vez entrenados, los LLM pueden ser usados para transformaciones entre DSL como alternativa a escribir una cadena de transformaciones de modelos si se aplica ingeniería dirigida por modelos o una transformación de programas en el contexto del grammarware. En nuestro caso, hemos creado modelos capaces de realizar conversiones bidireccionales para Orion y Athena.

Poder elegir como entrada entre escribir prompts en lenguajes natural o scripts del DSL. De este modo, se podría facilitar el uso de una aplicación a usuarios sin conocimientos de programación, algo que no siempre posibilita un DSL. Esta idea es la que se aplicó en [7] y en la que se han basado los creadores de DSL-Xpert [4]. En nuestro caso, Athena es sencillo para desarrolladores con experiencia en modelos de datos, y ahora se abre la puerta a que los esquemas de entrada a nuestras herramientas puedan ser escritos por expertos del dominio.

Prompts más simples para la generación de código. Cómo se supone en [1], los prompts para generar código pueden resultar más sencillos si el lenguaje destino es un DSL que si se debe generar código para varios frameworks o lenguajes de más bajo nivel como GPL o XML. En nuestro caso, una entrada en

Orion puede resultar más simple y precisa que especificar el cambio en lenguaje natural.

Facilitar el proceso de generación de código. El uso de un DSL eleva el nivel de abstracción y simplifica los prompts, disminuyendo la probabilidad de errores en la salida del LLM. Al existir validadores específicos (como en Athena y Orion), cualquier respuesta generada en forma de script de DSL puede verificarse automáticamente, agilizando la corrección de inconsistencias y garantizando mayor calidad en el proceso de generación de código.

También es preciso destacar que hay razones por las que conviene crear un DSL aunque ahora dispongamos de un nivel más alto de abstracción proporcionado por los LLMs, como los siguientes ejemplos. *Interoperabilidad*: Un DSL resulta más adecuado que el lenguaje natural para describir metadatos y garantizar la integración de herramientas, ya que reduce ambigüedades. *Familias de DSL*: A menudo se requiere un conjunto de lenguajes en lugar de uno solo [8], lo que complica su manejo exclusivo mediante un LLM.

Como trabajo futuro, se seguirá experimentando con otras operaciones como transformaciones de modelos, manejo de versiones, comparación de scripts y modelos, conversión bidireccional de gramáticas en metamodelos, refactoring de modelos, y trazabilidad en transformaciones, con el objetivo de crear un framework basado en LLM con operaciones básicas de manejo de modelos.

Agradecimientos Este trabajo es parte del proyecto PID2020-117391GB-I00 financiado por el Ministerio Español de Ciencia, Innovación y Universidades y la Agencia Estatal de Investigación (MICIU/AEI /10.13039/501100011033) y ERDF, EU.

Referencias

1. Bökelund, B.: How is genai different from other code generators? (2023), `https://martinfowler.com/articles/exploring-gen-ai.html`, accedido: 05-03-2025
2. Chillón, A.H., Klettke, M., Ruiz, D.S., Molina, J.G.: A generic schema evolution approach for nosql and relational databases. IEEE Trans. on Knowledge and Data Engineering **36**(7), 2774–2789 (2024)
3. Fernández, C.J., Sevilla, D., Molina, J.G.: A unified metamodel for nosql and relational databases. Information Systems **104**, 101898 (2022)
4. Garcia-Gonzalez, D., Lamas, V., R. Luaces, M.: Dsl-xpert: Llm-driven generic dsl code generation. In: Proc. of the ACM/IEEE 27th Int. Conf. on Model-Driven Engineering Languages and Systems. p. 16–20. MODELS Companion '24 (2024)
5. Hernández Chillón, A., Sevilla Ruiz, D., Garcia-Molina, J.: Athena: A Database-Independent Schema Definition Language. In: ER 2021 Workshops CoMoNoS, St.John's, NL, Canada. vol. 13012, pp. 33–42 (2021)
6. Mosthaf, M., Wasowski, A.: From a natural to a formal language with dsl assistant. In: Proc. of the ACM/IEEE 27th Int. Conf. on Model Driven Engineering, Languages and Systems. pp. 541–549 (2024)
7. Netz, L., Michael, J., Rumpe, B.: From natural language to web applications: using large language models for model-driven software engineering. In: Modellierung 2024. pp. 179–195 (2024)
8. Pérez-Berenguer, D., Molina, J.G.: Indieauthor: A metamodel-based textual language for authoring educational courses. IEEE Access **7**, 51396–51416 (2019)