Desarrollo Eficiente de Lenguajes Específicos de Dominio para la Ejecución de Procesos de Minería de Datos

Alfonso de la Vega, Diego García-Saiz, Marta Zorrilla, y Pablo Sánchez

Dpto. Ingeniería Informática y Electrónica
Universidad de Cantabria, Santander (España)
{alfonso.delavega, diego.garcia,
marta.zorrilla, p.sanchez}@unican.es

Resumen Aunque las técnicas de minería de datos están consiguiendo cada día una mayor popularidad, su complejidad impide que sean aún utilizables por personas sin un sólido conocimiento en las mismas. Una posible solución, ya explorada por los autores de este artículo, es la construcción de Lenguajes Específicos de Dominio que proporcionen una serie de primitivas de alto nivel para la ejecución de procesos de minería de datos. Dichas primitivas sólo hacen referencia a terminología propia del dominio analizado, enmascarando detalles técnicos de bajo nivel. No obstante, la construcción de un lenguaje específico de dominio puede ser un proceso costoso. Este artículo muestra cómo reducir los tiempos de desarrollo de estos lenguajes de análisis mediante la reutilización de partes comunes de estos DSLs.

Palabras clave: Lenguajes Específicos de Dominio · Desarrollo de Software Dirigido por Modelos · Minería de Datos

1. Introducción

El uso de técnicas de análisis de datos está adquiriendo cada vez una mayor popularidad. Este incremento se debe en parte a la mayor presencia en nuestra vida de sistemas informáticos, los cuales almacenan datos acerca de nuestras actividades que, convenientemente analizados, pueden ayudar a mejorar ciertos procesos y negocios. Un ejemplo de ello es cómo la plataforma de vídeo bajo demanda *Netflix* utiliza los datos de actividad de sus usuarios para decidir cuáles deberían ser las siguientes series y películas a producir [15].

No obstante, el análisis de estos datos es una tarea especializada que no está al alcance de cualquier profesional, ya que requiere de un conocimiento sólido y detallado de una serie de técnicas y algoritmos especializados. Prueba de ello es la creciente demanda de perfiles profesionales en *Ciencia de Datos*, los cuales son difíciles de encontrar y suelen tener un alto coste asociado [13].

Para solventar este problema, nos planteamos la idea de construir lenguajes de consulta, mediante primitivas de alto nivel y terminología propia de un dominio concreto. Estos lenguajes permitirían a las personas responsables de un

proceso especificar tareas que hicieran uso de técnicas de minería de datos, con el objetivo de extraer información que permitiese mejorar estos procesos, y haciendo innecesario que los usuarios deban poseer conocimientos acerca de dichas técnicas de minería.

En un trabajo anterior [16] exploramos la posibilidad de crear estos Lenguajes Específicos de Dominio (en adelante DSLs, por sus siglas en inglés *Domain Specific Languages*). Concretamente, se desarrolló un DSL para el ámbito educacional, que permitía analizar el rendimiento de un curso o asignatura a partir de datos recopilados por la plataforma de *e-learning* utilizada durante el desarrollo de dicho curso.

Aunque en [16] se demostró que es factible, bajo ciertas condiciones, crear este tipo de DSLs, se constató que el coste asociado a su desarrollo desde cero puede ser elevado. No obstante, en dicho trabajo se apreció la posibilidad de reutilizar ciertos elementos para el desarrollo de DSLs similares, lo que ayudaría a reducir sus tiempos de desarrollo y, por consiguiente, su coste.

Siguiendo esta idea, este trabajo presenta una infraestructura para el desarrollo de DSLs de análisis de datos, que trata de explotar dichas posibilidades de reutilización. El objetivo final es crear un marco de trabajo que permita desarrollar estos DSLs de una forma más eficiente que partiendo desde cero cada vez.

Para comprobar la efectividad de nuestra propuesta, se utilizó dicha infraestructura para generar dos DSLs: (1) el que se había creado manualmente para el ámbito educacional [16]; y (2) otro para el análisis de datos médicos relacionados con la diabetes. Se constató que era posible la reutilización de partes significativas de estos DSLs y que dicha reutilización ayudaba a reducir los tiempos asociados a su desarrollo.

Tras esta introducción, este artículo se estructura como sigue: La sección 2 describe la motivación existente tras este trabajo, así como trabajos previos relacionados que han intentado acercar la minería de datos a usuarios no expertos. La sección 3 describe la infraestructura creada para facilitar el desarrollo de DSLs para análisis de datos. Por último, la sección 4 concluye este trabajo, resaltando las principales conclusiones obtenidas y comentando posibles trabajos futuros.

2. Motivación y Antecedentes

Esta sección describe la motivación tras este trabajo. Primero, se describe por qué estamos interesados en desarrollar DSLs para el análisis de datos y en segundo lugar por qué es conveniente crear una infraestructura que facilite el desarrollo de dichos DSLs.

2.1. Por qué DSLs para Minería de Datos

El trabajo que aquí presentamos se basa en la siguiente hipótesis: las técnicas actualmente disponibles para el análisis de datos, y en especial las relacionadas

con la minería de datos, solamente pueden ser utilizadas por personas con un sólido conocimiento en las mismas. Si se desea que profesionales que carezcan de dicho conocimiento especializado utilicen o se beneficien de estas técnicas, es necesario el desarrollo de sistemas que oculten su complejidad.

Para verificar que esta hipótesis es cierta, realizamos una revisión sistemática de la literatura siguiendo las directrices de Kitchenham [9] y Wohlin [7]. En dicha revisión se analizaron una serie de herramientas y trabajos cuyo objetivo era acercar las técnicas de minería de datos a los usuarios. Una descripción detallada de los aspectos formales de dicha revisión sistemática, por razones de espacio, queda fuera del ámbito de este artículo. Como resultado de la misma, se identificaron cuatro grupos principales de trabajos y herramientas, los cuales describimos a continuación.

Soluciones Ad-Hoc para Problemas Concretos Este grupo se compone de aplicaciones diseñadas para resolver un problema de análisis de datos específico dentro de un dominio muy concreto. Por ejemplo, Kamsu et al. [8] desarrollaron un sistema para extraer patrones dentro de datos provenientes de hemodiálisis. En estos casos, el problema es bien conocido desde el inicio y las técnicas de minería de datos se seleccionan y optimizan para ese problema concreto. A continuación, se construye una interfaz amigable que permita a los usuarios, expertos en el dominio del problema pero no en técnicas de análisis de datos, ejecutar dichas técnicas. Dependiendo del trabajo, el usuario puede en cierta medida refinar los resultados obtenidos o modificar la tarea de análisis de datos.

El principal inconveniente de estos trabajos es su coste asociado, ya que implican desarrollar una aplicación desde cero. Además, las aplicaciones desarrolladas no son fácilmente adaptables a otros problemas o dominios.

Herramientas de Minería Basadas en Workflows Este conjunto agrupa una serie de herramientas para la definición de procesos de minería de datos mediante una serie de bloques reutilizables. Estos bloques representan algoritmos y tareas típicas de los procesos de minería de datos y se pueden combinar de forma rápida y eficiente para definir procesos completos de minería de datos. *Rapidminer* [1] y *Weka* [6] son ejemplos de este tipo de aplicaciones. Estas herramientas ayudan a reducir los tiempos de desarrollo de los procesos de minería, pero no están orientadas a usuarios no expertos. Por ejemplo, cada bloque requiere la correcta configuración de una serie de parámetros internos, no estando esa tarea al alcance del usuario medio, por lo que se precisa de la contratación de personal experto.

Metodologías Orientadas a Usuarios No Expertos Esta categoría contiene una serie de metodologías que tratan de asegurar que las aplicaciones desarrolladas puedan ser utilizadas por usuarios sin conocimientos de minería de datos.

Un ejemplo de estas metodologías es el trabajo de Zorrilla y García [17] donde se propone, en primer lugar, construir servicios que encapsulen procesos

de minería de datos para un dominio concreto. A diferencia de los bloques de la categoría anterior, se busca que estos servicios sean autoconfigurables [3]. A continuación, se desarrollarían una serie de interfaces web que permitirían a los usuarios no expertos seleccionar los conjuntos de datos a analizar, invocar dichos servicios y visualizar los resultados.

Estas metodologías siguen precisando de la contratación de expertos que las apliquen, aunque en muchos casos se intenta favorecer la reutilización de elementos. Además, en ciertas ocasiones, podría ser difícil adaptar las aplicaciones desarrolladas a nuevas situaciones. Por ejemplo, una aplicación para analizar datos de estudiantes podría no ser utilizable para analizar datos relativos a otras entidades del dominio, tales como exámenes o actividades en el aula.

Herramientas Genéricas Orientadas a Usuarios No Expertos Este grupo aglutina herramientas que proporcionan al usuario no experto comandos o primitivas de alto nivel que le permiten ejecutar ciertas tareas de minería de datos sin necesidad de poseer sólidos conocimientos sobre las mismas. En este grupo destacaríamos *Oracle Predictive Analytics* [4]. Esta herramienta proporciona una serie de procedimientos al estilo SQL que permiten extraer cierta información de los datos contenidos en una tabla de una base de datos (alojada en Oracle). Por ejemplo, el usuario puede utilizar el comando `EXPLAIN` para intentar averiguar qué valores de los atributos de la tabla producen un cierto valor en una columna determinada. De este modo, usando este comando se podría averiguar por qué ciertos estudiantes tienen la columna `Calificación` con el valor `Suspenso`.

Una ventaja inicial es que no es necesaria la contratación de expertos para ejecutar estos comandos (aunque podrían requerirse otras cuestiones, como familiaridad con Oracle, por ejemplo). La gran ventaja de esta técnica es que estos comandos no precisan ser modificados cuando se utilizan para nuevos datos. Por ejemplo, el comando `EXPLAIN` es el mismo con independencia de la tabla sobre la que se aplique. Por tanto, estos comandos sí podrían aplicarse a datos de estudiantes, exámenes o actividades en aula, siempre que estuviesen disponibles en sus correspondientes tablas. Sin embargo, esta genericidad, tiene una penalización sobre la precisión de los resultados, ya que las técnicas subyacentes utilizadas no están convenientemente optimizadas para cada dominio.

Tras analizar el estado del arte, decidimos explorar la siguiente idea: En un primer lugar, encapsularíamos diversos procesos de minería de datos en una serie de componentes reutilizables. El objetivo inicial era que estos componentes fuesen reutilizables para problemas y datos procedentes de diversos dominios, al estilo de *Oracle Predictive Analytics*. No obstante, estos componentes deberían poderse optimizar para un problema concreto si así se desease, tal como en Zorrilla y García [17]. En segundo lugar, para que estos componentes fuesen utilizables por usuarios no expertos, desarrollaríamos DSLs, al estilo de *Oracle Predictive Analytics*, pero adaptados a las necesidades de cada usuario. Estos DSLs tendrían que permitir plantear un problema de minería de datos utilizando primitivas de alto nivel y una jerga cercana al dominio del usuario. Se busca

```
00 show_profile of Students;
01 show_profile of Students with courseOutcome=fail;
02 find_reasons_for courseOutcome=fail of Students;
```

Figura 1. Ejemplo de consultas para el ámbito educacional.

además que la forma de plantear dichos problemas sea lo más flexible posible. Las sentencias escritas utilizando el DSL se transformarían en código de bajo nivel encargado de invocar los componentes reutilizables para dar respuesta al problema planteado por el usuario.

Esta idea fue inicialmente explorada mediante la creación de un DSL, más un conjunto de procesos de minería de datos, para el ámbito educacional. La siguiente sección describe dicho trabajo.

2.2. Un DSL para el Análisis de Datos Educacionales

Para explorar nuestra idea inicial, desarrollamos un DSL para el análisis de datos provenientes del dominio educacional [16]. El objetivo de este DSL era que los profesores pudiesen explotar los datos que plataformas de aprendizaje como Moodle [11] recopilan acerca del desarrollo de sus asignaturas, con el objetivo de extraer de dichos datos información que pudiese ayudar a mejorar el rendimiento las asignaturas.

Concretamente, tratamos de resolver dos tareas de análisis. La primera de ellas consistía en identificar los diferentes tipos de perfiles que existían dentro de un conjunto de datos. Este análisis permitiría, por ejemplo, dividir a los estudiantes de un curso en grupos con características comunes.

La segunda de ellas trataba de encontrar las causas por las cuales se producía cierto fenómeno. Por ejemplo, por qué los estudiantes no superaban el curso, por qué lo abandonaban o por qué dejaban de entregar una determinada tarea, entre otras cuestiones.

Para ambas tareas, creamos código Java que utilizaba una serie de algoritmos de minería de datos proporcionados por la plataforma *Weka* [6]. Este código se parametrizó para que pudiese ser generado mediante plantillas.

A continuación, se desarrolló un DSL para invocar dichos procesos reutilizables. La Figura 1 muestra ejemplos de consultas escritas en dicho DSL[1].

La primera consulta (Figura 1, Línea 00) permite al profesor dividir el conjunto de estudiantes de su curso en grupos que compartan características similares. Para ello aplica la primitiva `show_profile` a la entidad `Students`.

La segunda consulta (Figura 1, Línea 01) sería similar a la primera, pero restringe los datos de entrada, aplicando un filtro, a aquellos estudiantes que no hayan superado la asignatura. La idea en este caso es caracterizar grupos de estudiantes que no superan la asignatura.

[1] El DSL se presentó inicialmente en un foro de ámbito internacional, y es por ello por lo que sus términos están en inglés.

La tercera consulta (Figura 1, Línea 02) trata de encontrar las causas por las que los estudiantes no superan el curso. Para ello hace uso de la primitiva `find_reasons_for`, la cual requiere que se le proporcione un fenómeno a explicar (`courseOutcome=fail`) y una entidad (`Students`) a analizar.

Este DSL se implementó definiendo su correspondiente metamodelo en Ecore y su sintaxis textual con ayuda de *Xtext* [5]. La semántica quedaba definida mediante la generación, a partir de las consultas, de código Java que invocaba los correspondientes algoritmos de la plataforma Weka. La generación de código se implementó utilizando las herramientas facilitadas por la suite de ingeniería de modelos *Epsilon* [12].

Como puede apreciarse, el DSL sólo contiene una serie de primitivas de alto nivel (`find_reasons_for`, `show_profile`) y referencias a entidades procedentes del dominio del usuario (`Student`) o a atributos de dichas entidades (`courseOutcome`).

A partir del nombre de una entidad, el lenguaje debería ser capaz de recuperar el conjunto de datos a analizar. Por tanto, debe existir una correspondencia entre estos nombres y los conjuntos de datos disponibles para su análisis. En nuestro caso, dicho nombre debe hacer referencia al nombre de un fichero de datos en formato ARFF, que es el formato que utiliza la plataforma Weka [6]. Cómo se obtiene la información a procesar y se almacena en dichos ficheros ARFF queda fuera del ámbito de este artículo.

Para fomentar su usabilidad, el DSL creado trata de minimizar los errores que pudiese cometer el usuario a la hora de escribir una consulta. Ello requiere: (1) que se eviten tanto como sea posible elementos libres; y (2) que se ofrezcan tantas funciones de autocompletado o asistencia como sea necesario. Por ejemplo, tal como se explicó anteriormente, como nombre de una entidad no debería ser posible escribir cualquier cadena, ya que este nombre debe corresponderse con el de un fichero de datos disponible. Por tanto, el DSL debería: (1) verificar esta restricción antes de compilar la consulta; y (2) ofrecer el conjunto de opciones disponibles al usuario durante la especificación de una consulta.

2.3. Problemas a Resolver

Tras la realización de dicho trabajo se identificaron dos problemas a resolver: (1) resultaba difícil en determinados contextos crear procesos de minería de datos que fuesen realmente reutilizables para diferentes dominios; y (2) el desarrollo de un DSL como éste desde cero puede convertirse en un proceso costoso que requeriría la contratación de expertos en DSLs. En este caso, estaríamos simplemente reemplazando el problema de tener que contratar expertos en minería de datos por el problema de tener que contratar expertos en el desarrollo de DSLs.

Este trabajo se centra en la resolución del segundo de estos problemas. El primer problema es un problema recurrente en minería de datos, que obliga a que, en muchas ocasiones, si se quieren obtener resultados precisos, los procesos tengan que ser diseñados ad-hoc para resolver un problema concreto y bien determinado. Dicho problema queda fuera del ámbito de este artículo, y será objeto de estudio en trabajos futuros. Por tanto, en adelante, asumiremos como

hipótesis que existen procesos de minería de datos reutilizables para diferentes dominios.

Con relación al segundo problema, se advirtió que ciertas partes de este DSL podían ser fácilmente reutilizables para el desarrollo de DSLs para nuevos dominios. Por ejemplo, la sintaxis textual concreta sería prácticamente la misma, ya que de dominio a dominio simplemente variarían los nombres que pueden ser utilizados como entidades del dominio. Igualmente, gran parte del proceso de generación de código debería ser reutilizable.

La solución presentada en este trabajo trata de explotar esta idea para crear un ecosistema de generación de estos DSLs para el análisis de datos. Para ello se ha rediseñado el DSL educacional, creando una infraestructura que permita un desarrollo más eficiente de DSLs para otros dominios. Dicha infraestructura se ha utilizado para el desarrollo de un DSL para el análisis de datos relacionados con pruebas de diabetes. Dicho dominio se describe a continuación.

2.4. Ejemplo: Análisis de Factores Relacionados con la Diabetes

Para el desarrollo del nuevo DSL se ha utilizado un conjunto de datos procedentes de pruebas realizadas para comprobar la existencia de diabetes en pacientes [14]. Por cada paciente, se recogen una serie de datos así como el resultado final de la prueba (positivo o negativo). Estos datos se pueden analizar para intentar averiguar si existen factores que puedan ser considerados como causas de una diabetes. Es decir, querríamos poder escribir una consulta en nuestro DSL como `find_reasons_for test_result = positive of Diabetes_Results`.

De igual forma, podría ser interesante plantear otras consultas, como averiguar si existen diferentes grupos o perfiles de pacientes con diabetes.

La siguiente sección describe cómo se ha creado la infraestructura anteriormente mencionada y cómo se ha utilizado para construir el DSL para el análisis de los datos de diabetes.

3. Una Infraestructura para el Desarrollo de DSLs para Minería de Datos

Para desarrollar una infraestructura que nos permitiese crear de manera eficiente un ecosistema de DSLs, en primer lugar precisábamos identificar qué elementos de dichos DSLs podían ser reutilizables.

Este proceso lo realizamos en dos fases: primero analizamos los componentes relacionados con la gramática y el editor del DSL; y luego aquellos relacionados con la generación del código encargado de procesar las consultas. Estos procesos se describen a continuación.

3.1. Desarrollo del Editor para el DSL

La gramática de nuestra familia de DSLs posee una componente claramente común. Esta componente abarca el conjunto de comandos o primitivas, tales

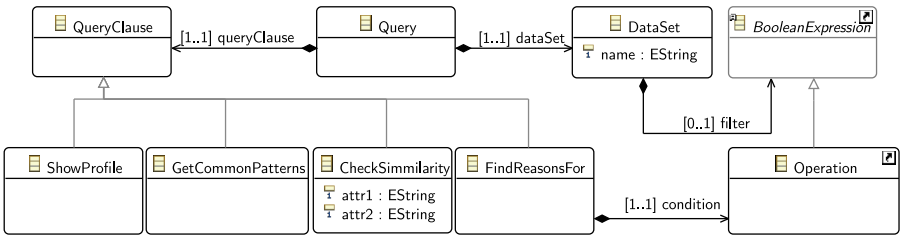

Figura 2. Sintaxis abstracta genérica.

como `find_reasons_for`, que el usuario puede utilizar para invocar tareas de análisis de datos. Por otro lado, existe una parte variable, que es la correspondiente a las referencias a entidades del dominio del usuario.

Estas referencias estaban incluidas en la gramática por dos razones: (1) evitar que el usuario pueda especificar consultas que se refieran a entidades inexistentes para las cuales no tenemos datos; (2) poder utilizar una serie de funciones de autocompletado que facilitasen al usuario la especificación de las consultas.

Teniendo estos factores en cuenta, la estrategia que adoptamos fue la siguiente: En primer lugar, se rediseñó la gramática para que fuese genérica para cualquier dominio. Con este fin, se liberó a la gramática de la responsabilidad de asegurar que las referencias a las entidades del dominio fuesen correctas. Esta responsabilidad se delegó en un *validador* externo, que se encargaría de asegurar que se satisfacen las restricciones relativas a las entidades del dominio. Para ello, el validador utilizaría como entrada un modelo externo a la gramática que especificase qué entidades y atributos están disponibles dentro de cada dominio. Las funciones de asistencia y autocompletado también harían uso de este modelo de entidades. De esta forma, para adaptar el DSL a un nuevo dominio, sólo sería necesario proporcionar un nuevo modelo de entidades y regenerar el editor.

La Figura 2 muestra parte de la sintaxis abstracta genérica creada para nuestra familia de DSLs. En ella se aprecia que cada consulta (`Query`) está definida por una primitiva (`QueryClause`) que determina la tarea de minería de datos a realizar. Esta consulta actúa sobre un determinado conjunto de datos (`Dataset`) el cual, de acuerdo con esta gramática, puede quedar definido por una cadena de caracteres arbitraria. Esto permite que la gramática sea utilizable para cualquier dominio. No obstante, utilizando sólo esta gramática, se podrían especificar consultas que no tuviesen ninguna relación con los conjuntos de datos disponibles.

Este problema es evitado con la ayuda del validador externo introducido que lleva a cabo estas comprobaciones. Para realizar esta comprobaciones, el validador utiliza un modelo de entidades, que debe ser conforme a un metamodelo de entidades. Dicho metamodelo se muestra en la Figura 3.

De acuerdo a este metamodelo, cada entidad tiene un nombre y un conjunto de atributos. Estos atributos pueden ser de diferentes tipos. Por ejemplo, pueden ser numéricos (`NumericalAttribute`) o enumerados (`NominalAttribute`). Los

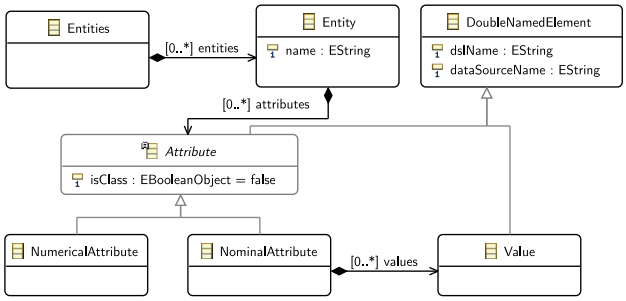

Figura 3. Metamodelo para la definición de información de las entidades analizadas.

```
01  @Check
02  def checkDataSetName(DataSet dataset) {
03    if (!entitiesProvider.entities.exists[
04          entity | entity.name.equals(dataset.name)
05        ]) {
06      error("Dataset with name '" + dataset.name + "' does not exist",
07          EdmdslPackage::eINSTANCE.dataSet_Name)
08    }
09  }
```

Figura 4. Función para validar el nombre de un `Dataset`.

atributos enumerados poseen asociado el conjunto finito de valores que pueden adoptar.

Además, cada atributo puede ser considerado como de clase (`isClass = true`) si sirve para particionar un conjunto de datos en clases relevantes para algún propósito. Esta distinción es relevante ya que en ciertas partes de nuestro DSL sólo se pueden utilizar atributos considerados de clase. Por ejemplo, puede interesar preguntar las causas por las que ciertos pacientes tienen diabetes (`find_reasons_for test_result=positive of Diabetes_Result`), pero no tendría mucho sentido preguntar las causas por las que un paciente reside en Londres (`find_reasons_for live_in=LONDON of Diabetes_Result`).

El validador externo se desarrolló utilizando las facilidades proporcionadas por Xtext. Para ello, especificamos una serie de funciones auxiliares en Xtend. A modo de ejemplo, la Figura 4 muestra la función que comprueba que el nombre de un `Dataset` introducido se corresponde con el nombre de una entidad de nuestro dominio. Como se puede observar, la función accede al modelo de entidades a través del objeto `entitiesProvider` (Figura 4, Línea 03) y comprueba si existe al menos una entidad cuyo nombre sea igual al del dataset.

De igual forma, para proporcionar funciones de asistencia y autocompletado, definimos otra serie de funciones auxiliares en Xtend. A modo de ejemplo, la Figura 5 muestra la función que carga los nombres de las entidades en el correspondiente menú contextual. Gracias a esta función, cada vez que el usuario

```
01  override public void completeDataSet_Name(EObject model,
02      Assignment assignment, ContentAssistContext context,
03      ICompletionProposalAcceptor acceptor) {

04    for (entity : entitiesProvider.entities) {
05      acceptor.accept(createCompletionProposal(entity.name, context))
06    }
07  }
```

Figura 5. Función que muestra los nombres disponibles para especificar un `Dataset`.

necesite escribir el nombre de un `Dataset`, aparecerá un listado con el conjunto de entidades disponibles.

Merece la pena destacar que estas funciones auxiliares de validación y asistencia al usuario serían reutilizables para diferentes dominios. Es decir, cada vez que queramos generar un DSL para un nuevo dominio, sólo necesitaríamos proporcionar el correspondiente modelo de entidades. El resto de elementos requeridos para el desarrollo del editor del DSL (sintaxis abstracta, sintaxis concreta, funciones auxiliares de validación, funciones auxiliares de asistencia y código de infraestructura) permanecerían sin modificar. Esto permite reducir los tiempos de desarrollo asociados al desarrollo de DSLs para nuevos dominios.

3.2. Desarrollo de los Generadores de Código

Tal como se comentó en la Sección 2.2, para ejecutar las consultas, éstas se transforman en código Java que invoca una serie de algoritmos de minería de datos proporcionados por la plataforma *Weka*. El proceso de transformación es completamente invisible para el usuario final, abstrayéndose por tanto de los detalles de bajo nivel. En este proceso, si se desea obtener mejores resultados o dependiendo las características de cada dominio, podría ser necesario modificar los algoritmos de minería de datos utilizados.

Por ejemplo, en base al dominio, para ejecutar la primitiva *show_profile* podría ser más adecuado utilizar como técnica de *clustering* el algoritmo *K-means*, mientras que en otros dominios el algoritmo *EM* (*Expectation Maximization*) podría generar mejores resultados.

Por otra parte, y aunque de momento no ha sido necesario, podría variar la plataforma de destino que proporciona las implementaciones de los algoritmos de minería de datos. Por ejemplo, podríamos optar por utilizar librerías presentes en *R* [2] en lugar de Weka.

Para ayudar a soportar esta variabilidad, introdujimos un metamodelo intermedio en el proceso de transformación para representar procesos de minería de datos de manera abstracta e independiente de cualquier plataforma. La parte izquierda de la Figura 6 muestra un fragmento de este metamodelo. Cada procedimiento se corresponde con un algoritmo de minería de datos que podría ser ejecutado. Estos algoritmos podrían requerir la configuración de parámetros adicionales. Por ejemplo, para llevar a cabo una tarea de clasificación, es necesario saber con respecto a qué atributo de un conjunto de datos deseamos realizarla

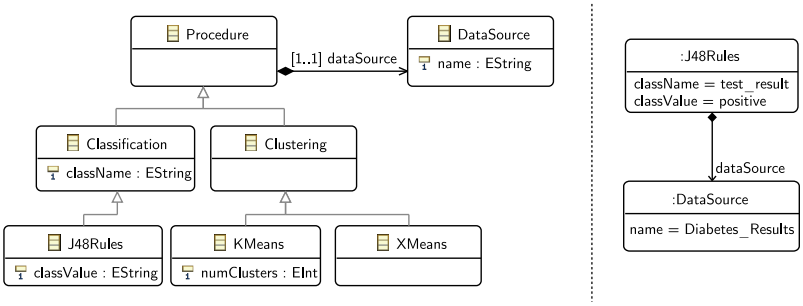

Figura 6. Izquierda: Metamodelo de un proceso abstracto de minería de datos; derecha: modelo del proceso de minería resultante de la transformación M2M.

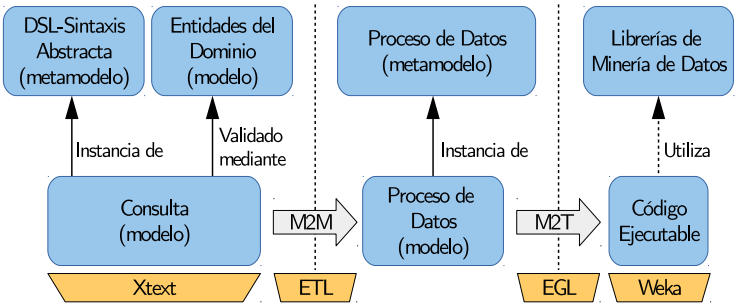

Figura 7. Transformaciones del proceso de ejecución de una consulta del DSL, indicándose en la parte inferior las herramientas involucradas en cada una de las etapas.

(atributo `className`). En el caso de querer utilizar el proceso *J48Rules*, que permite obtener las reglas que tratan de explicar por qué el atributo seleccionado toma un valor determinado, deberemos especificar además dicho valor de ese atributo (`classValue`).

Usando este metamodelo, una consulta de nuestro DSL se transforma, en primer lugar, en un modelo independiente de la plataforma que especifica qué algoritmos de minería de datos deben ser ejecutados para dar respuesta a esa consulta, tal como muestra la Figura 7. En segundo lugar, se realiza la generación de código para una plataforma concreta a partir de este modelo intermedio. La primera transformación modelo a modelo se realiza con el lenguaje *ETL (Epsilon Transformation Language)*, mientras que para la generación de código se emplea el lenguaje basado en plantillas *EGL (Epsilon Generation Language)*. Este proceso de transformación se ilustra a continuación para el ejemplo de los datos de diabetes.

Continuando con nuestro ejemplo anterior, la consulta `find_reasons_for test_result=positive of Diabetes_Result`, ya escrita en el DSL, se transformaría en el modelo mostrado en la parte derecha de la Figura 6. Este modelo indica que, para encontrar las razones por las cuales ciertos pacientes dan positivo en diabetes, se computará un árbol de decisión utilizando el algoritmo J48 [10]. Se

```
   // data obtention
01 DataSource source = new DataSource("data/diabetesResults.arff");
02 Instances ins = source.getDataSet();
03 ins.setClass(ins.attribute("test_result"));
   // rules generation
04 J48 j48 = new J48();
05 j48.buildClassifier(ins);
06 RuleSet ruleSet = j48.toRules();
07 ruleSet = ruleSet.filterByConsequent("positive");
   // results visualization
08 RulesVisualizer.show(ruleSet);
```

Figura 8. Código resultante de la transformación M2T sobre el modelo de minería.

```
IF (body_mass_index > 29.9 AND
    plasma_glucose_concentration > 157)
THEN result = tested_positive

Support: 11.979%, Confidence: 86.957%
```

Figura 9. Ejemplo del tipo de reglas obtenidas mediante la ejecución de la consulta.

analizarán los datos de los pacientes contenidos en el dataset `Diabetes_Result` utilizando como clase el atributo `test_result` para la construcción del árbol de decisión. A continuación, el algoritmo extraerá aquellas ramas que sirvan para explicar por qué ciertos pacientes padecen diabetes, es decir, aquellas cuyo valor predicho sea `positive`.

Una vez generado el modelo intermedio, (Figura 6, derecha), éste se usa como entrada para las plantillas de generación de código. En nuestro caso, las plantillas convertirían el modelo en código Java. La Figura 8 muestra parte del código generado. En primer lugar, se obtienen los datos mediante la carga del dataset especificado en un objeto de la clase *Instances* (Figura 8, Líneas 01-02). Esta clase Instances es una clase proporcionada por la plataforma Weka. Posteriormente, se ejecuta el algoritmo *J48*, tomando *test_result* como atributo de clase (Figura 8, Líneas 03-05). Dado que sólo nos interesan las reglas que retornen *positive* como resultado, aplicamos un filtro para quedarnos con aquellas reglas cuyo consecuente sea igual a *positive* (Figura 8, Líneas 06-07). Finalmente, se muestra las reglas obtenidas por pantalla (Figura 8, Línea 08).

La Figura 9 muestra, a modo de ejemplo, una de las reglas que el sistema mostraría como resultado al usuario que ha realizado la consulta para un dataset concreto. Esta regla determina que una causa de la diabetes puede ser poseer un índice de masa corporal superior a 29.9 y una concentración de glucosa en plasma mayor de 157. Además, para especificar el grado de certeza que podemos atribuir en dicha regla, se indica que este patrón aparecía en un 12 % de los casos y que, dentro de esos casos, prácticamente un 87 % padecía diabetes. Es decir, el patrón se da en un sector reducido de la población analizada, pero cuando aparece podría ser causa de una posible diabetes.

El uso del metamodelo intermedio proporciona las siguientes ventajas:

1. La introducción de cambios en el DSL afectaría solamente al proceso de transformación entre modelos, pero no a las plantillas de generación de código, siempre y cuando el cambio en el DSL no implicase la incorporación de nuevos elementos en el metamodelo de procesos de minería. Esto nos permite realizar pequeños cambios en la sintaxis del DSL sin tener que modificar las plantillas de generación de código.
2. Para cambiar el algoritmo de minería utilizado para computar una primitiva del DSL, sólo es necesario cambiar las transformaciones modelo a modelo ejecutadas. No sería necesario modificar las plantillas de generación de código, siempre que el nuevo algoritmo utilizado ya exista en el metamodelo de procesos de minería.
3. Si deseásemos cambiar la plataforma destino utilizada para ejecutar los algoritmos de minería de datos, sólo sería necesario modificar las plantillas de generación de código, permaneciendo las transformaciones modelo a modelo inalteradas.
4. El proceso de traducción de una consulta en un algoritmo de minería de datos se simplifica, ya que las transformaciones modelo a modelo están libres de la complejidad accidental introducida por las plataformas destino.

4. Sumario y Trabajos Futuros

La democratización de la minería de datos requiere del desarrollo de sistemas que permitan a usuarios no expertos beneficiarse de estas técnicas. Para alcanzar dicho objetivo, propusimos en un trabajo previo [16] la utilización de DSLs para minería de datos. Aunque los resultados iniciales eran prometedores, existían varios problemas a resolver. Uno de estos problemas era el coste asociado al desarrollo de un DSL.

Para aliviar dicho problema, en este trabajo se ha presentado una infraestructura de asistencia al desarrollo de estos DSLs. Esta infraestructura permite que se puedan reutilizar partes importantes de otros lenguajes de análisis de datos para la construcción de un DSL en un nuevo dominio. De igual forma, la infraestructura trata de minimizar los impactos producidos a la hora de introducir ciertos cambios en los DSLs. Esto permite reducir los tiempos de desarrollo de nuevos DSLs, reduciendo a la vez su coste.

Como trabajos futuros, se seguirá explorando la viabilidad de esta idea mediante la incorporación de nuevos dominios y/o problemas de análisis. Además, se estudiará la posibilidad de generar ciertos elementos de esta infraestructura. Por ejemplo, el modelo de entidades debería poderse generar sin demasiada dificultad desde los conjuntos de datos disponibles.

Agradecimientos. Este trabajo ha sido parcialmente financiado por el Gobierno de Cantabria (España) mediante el Programa de Personal Investigador en Formación Predoctoral de la Universidad de Cantabria y por el Gobierno de España en el proyecto TIN2014-56158-C4-2-P(M2C2).

Referencias

1. Rapidminer. https://rapidminer.com/
2. The R Project for Statistical Computing. https://www.r-project.org/
3. Balcázar, J.L.: Parameter-free Association Rule Mining with Yacaree. In: Actes Extraction et Gestion des Connaissances (EGC). pp. 251–254. Brest (France) (January 2011)
4. Campos, M., Stengard, P., Milenova, B.: Data-Centric Automated Data Mining. In: Fourth International Conference on Machine Learning and Applications (ICMLA'05). vol. 2005, pp. 97–104 (2005)
5. Eysholdt, M., Behrens, H.: Xtext: Implement Your Language Faster than the Quick and Dirty Way. In: Companion to the 25th Annual Conference on Object-Oriented Programming, Systems, Languages, and Applications (SPLASH/OOPSLA). pp. 307–309. Reno/Tahoe (Nevada, USA) (October 2010)
6. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA Data Mining Software: An Update. SIGKDD Explorations Newsletter 11(1), 10–18 (June 2009)
7. Jalali, S., Wohlin, C.: Systematic literature studies: database searches vs. backward snowballing. In: Proceedings of the 2012 6th ACM_IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). pp. 29–38 (2012)
8. Kamsu-Foguem, B., Tchuenté-Foguem, G., Allart, L., Zennir, Y., Vilhelm, C., Mehdaoui, H., Zitouni, D., Hubert, H., Lemdani, M., Ravaux, P.: User-centered visual analysis using a hybrid reasoning architecture for intensive care units. Decision Support Systems 54(1), 496–509 (2012)
9. Kitchenham, B., Charters, S.: Guidelines for performing Systematic Literature Reviews in Software Engineering. Tech. Rep. EBSE 2007-001, Keele University and Durham University Joint Report (2007)
10. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1993)
11. Rice, W.: Moodle E-Learning Course Development. Packt Publishing (2006)
12. Rose, L.M., Paige, R.F., Kolovos, D.S., Polack, F.A.C.: Model Driven Architecture – Foundations and Applications: 4th European Conference, ECMDA-FA 2008, Berlin, Germany, June 9-13, 2008. Proceedings, chap. The Epsilon Generation Language, pp. 1–16. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
13. Schrage, M.: Stop Searching for That Elusive Data Scientist. Harvard Business Review, https://hbr.org/2014/09/stop-searching-for-that-elusive-data-scientist/
14. Smith, J.W., Everhart, J.E., Dickson, W.C., Knowler, W.C., Johannes, R.S.: Using the ADAP Learning Algorithm to Forecast the Onset of Diabetes Mellitus. Proceedings of the Annual Symposium on Computer Application in Medical Care pp. 261–265 (nov 1988), http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2245318/
15. Sweney, M.: Netflix gathers detailed viewer data to guide its search for the next hit. The Guardian, http://www.theguardian.com/media/2014/feb/23/netflix-viewer-data-house-of-cards
16. de la Vega, A., García-Saiz, D., Zorrilla, M., Sánchez, P.: Towards a DSL for Educational Data Mining. In: Sierra-Rodríguez, J.L., Leal, J.P., Simões, A. (eds.) Languages, Applications and Technologies SE - 8, Communications in Computer and Information Science, vol. 563, pp. 79–90. Springer International Publishing (2015)
17. Zorrilla, M., García-Saiz, D.: A service-oriented architecture to provide data mining services for non-expert data miners. Decision Support Systems 55(1), 399 – 411 (2013)