ScheMoL: Un lenguaje específico del dominio para extraer modelos de bases de datos relacionales

Javier Luis Cánovas Izquierdo[1], Óscar Díaz[2], Gorka Puente[2], and Jesús García Molina[1]

[1] Grupo ModelUM, Universidad de Murcia, {jlcanovas,jmolina}@um.es
[2] Grupo ONEKIN, Universidad del País Vasco, {oscar.diaz,gorka.puente}@ehu.es

Abstract. La Modernización Dirigida por Modelos ha emergido recientemente como una nueva área dedicada a la automatización basada en modelos de procesos de reingeniería o modernización de software. En estos procesos existe una primera etapa de ingeniería inversa en la cual se aplican tareas de extracción de modelos a partir de los artefactos del sistema (p. ej., código o datos). En esta demostración presentamos ScheMoL, un lenguaje específico del dominio para extraer modelos desde datos almacenados en una base de datos relacional. El lenguaje permite definir transformaciones datos-a-modelo basadas en reglas que están inspiradas en las reglas del lenguaje de transformación modelo-a-modelo ATL. En el caso de ScheMoL, las reglas tienen como elemento origen una tabla y como elemento destino una metaclase de un metamodelo. También incorpora un lenguaje de consultas especialmente adaptado para obtener información de las bases de datos.

1 Introducción

Las técnicas del paradigma del Desarrollo de Software Dirigido por Modelos (DSDM) no son sólo aplicables para crear nuevos sistemas software sino también pueden ser utilizadas en tareas de reingeniería o modernización de software [1]. Así, la Modernización de Software Dirigida por Modelos ha emergido como una nueva disciplina donde los modelos guían todo el proceso de reingeniería.

En un proceso basado en DSDM, los desarrolladores definen inicialmente los modelos del sistema que posteriormente son transformados para crear determinadas partes del sistema software. Sin embargo, en un proceso de reingeniería o modernización, se parte de un sistema software ya existente y es necesario que los modelos iniciales del proceso se obtengan a partir de los artefactos software para entonces aplicar las técnicas del DSDM. Existe por tanto una primera fase de ingeniería inversa encargada de extraer obtener modelos de alto nivel de abstracción a partir de los artefactos del sistema software origen. A continuación, durante la fase de reestructuración, estos modelos extraídos son transformados en otros que conformen a la arquitectura destino aplicando transformaciones modelo-a-modelo (*m2m*). Finalmente, en la fase de ingeniería directa, estos modelos que conforman a la arquitectura destino son transformados para generar los artefactos del nuevo sistema por medio de transformaciones modelo-a-texto (*m2t*). En la actualidad, existen diferentes lenguajes de transformación *m2m* [2,

3] y de transformación *m2t* [4, 5] para realizar las tareas requeridas en las fases de reestructuración e ingeniería directa, respectivamente. Sin embargo, no existen lenguajes especialmente diseñados para definir el proceso de extracción de modelos, el cual es normalmente implementado haciendo uso de lenguajes GPL.

La extracción de modelos a partir de código fuente y datos es una tarea necesaria en cualquier modernización basada en modelos puesto que son los elementos principales de cualquier sistema software. Mientras que existen diferentes enfoques para abordar la extracción de modelos desde el código fuente [6, 7], todavía no se han planteado soluciones para la ingeniería inversa de datos utilizando el DSDM. De hecho, la extracción de modelos de los datos se realiza actualmente por medio de soluciones *ad hoc* o utilizando *mappers* objeto relacionales que requieren un gran esfuerzo de implementación (en [8] se realiza un estudio detallado de las principales soluciones existentes). Con estos problemas en mente, decidimos crear ScheMoL, un lenguaje específico del dominio (*Domain Specific Language*, DSL) para extraer modelos desde bases de datos relacionales.

A continuación se presentan las características principales del lenguaje ScheMoL y luego un ejemplo de transformación.

2 ScheMoL

En ScheMoL, el proceso de extracción de modelos es considerado como una transformacion datos-a-modelo (*d2m*) dirigida por una definición de transformación que define las correspondencias entre los elementos del esquema de la base de datos (tuplas) y los del metamodelo (metaclases). Un proceso de transformación de ScheMoL tiene como entradas los datos que conforman al esquema, el metamodelo al cual debe conformar el modelo resultante y la definición de transformación.

El lenguaje está profundamente inspirado en el lenguaje Gra2MoL [6], que es un lenguaje para definir transformaciones código-a-modelo. ScheMoL comparte con Gra2MoL el hecho de utilizar reglas así como el concepto de *binding*, el cual fue adaptado para tratar con tuplas. Además, ScheMoL también incluye un lenguaje de consultas adaptado para facilitar la obtención de información de bases de datos para inicializar las propiedades de los elementos del modelo.

Una definición de transformación ScheMoL está compuesta por un conjunto de reglas y, opcionalmente, un preámbulo. Cada regla de ScheMoL define las correspondencias entre una tabla del esquema de la base de datos y una metaclase del metamodelo y tiene cuatro secciones principales (ver Figura 1b):

- La sección *from*, que especifica la tabla origen junto con una variable que se asociará a la tupla cuando la regla sea ejecutada.
- La sección *to*, que especifica la metaclase destino junto con una variable que se asociará a la instancia de dicha metaclase cuando la regla sea ejecutada.
- La sección *filter* es opcional e incluye una expresión condicional que se aplica a la tupla origen de forma que solamente aquellas tuplas que satisfagan dicha condición ejecutarán la regla.
- La sección *mapping*, que contiene un conjunto de *bindings* para establecer las propiedades del elemento del metamodelo destino.

El preámbulo de una transformación ScheMoL permite especificar las claves primarias/ajenas de las tablas así como vistas de la base de datos. En primer lugar, las claves primarias/ajenas pueden ser especificadas explícitamente en aquellos casos en los que la base de datos no incluya su definición (p. ej., en MySQL), permitiendo que el lenguaje de consultas pueda consultar la base de datos de forma transparente. Por otro lado, las vistas permiten adaptar la base de datos para facilitar la definición de la transformación. Una definición de vista tiene el mismo formato que una vista en SQL y se utiliza de forma transparente desde el lenguaje de consultas como una tabla más.

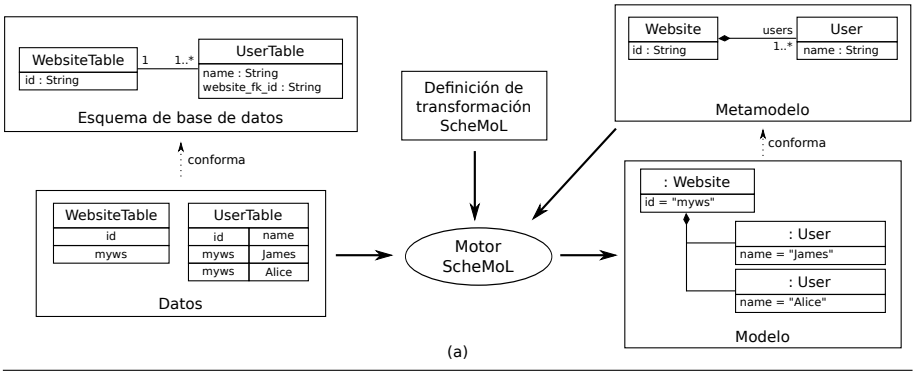
La ejecución de una transformación ScheMoL está guiada por los *bindings*, que tienen una sintaxis y semántica muy similares a las utilizadas Gra2MoL y en el lenguaje de transformación modelo-a-modelo ATL [2]. En ScheMoL, la parte izquierda del *binding* debe ser una propiedad de la metaclase destino de la regla pero la parte derecha puede ser un valor literal, una expresión o una consulta, donde las dos últimas pueden tratar con tuplas. Las consultas se expresan utilizando un lenguaje especialmente adaptado que permite obtener información de la base de datos sin necesidad de escribir consultas complejas en SQL, tal y como se ha comentado anteriormente. En [8] se describe detalladamente el mecanismo de ejecución de *bindings* así como el lenguaje de consultas.

En lenguaje fue utilizado en diferentes casos de estudio para extraer modelos a partir de bases de datos utilizadas por aplicaciones Web 2.0. El desarrollo de estos casos de estudio permitió validar el lenguaje así como incorporar un mecanismo de extensión para la definición de nuevos operadores en la sección de *mapping* (p. ej., operadores para tratar con datos anotados).

3 Ejemplo

La Figura 1a muestra un ejemplo simple de un proceso de transformación basado en ScheMoL para la extracción de modelos a partir de una base de datos de usuarios de un portal web. Las entradas del proceso son: (1) el esquema de la base de datos, que contiene la tabla `WebsiteTable` y `UserTable`, donde la tabla `UserTable` tiene una clave ajena a la tabla `WebsiteTable` (columna `website_fk_id`); (2) un conjunto de datos que conforman al esquema de la base de datos; (3) un metamodelo simple para representar sitios web (metaclase `Website`) y sus usuarios (metaclase `User`); y (4) la definición de transformación de ScheMoL, que se explica a continuación. El resultado del proceso de transformación es un modelo que conforma con el metamodelo destino.

La definición de transformación de este ejemplo está compuesta de dos reglas (`mapWebsite` y `mapUser`, ver Figura 1b). La regla `mapWebsite` define la correspondencia entre la tabla `WebsiteTable` y la metaclase `Website`. Esta regla inicia la ejecución de la transformación y crea una instancia de la metaclase `Website` a partir de la única tupla de la tabla `WebsiteTable` del ejemplo. El primer *binding* ejecuta la consulta `wTab.id`, la cual obtiene el valor de la columna `id` para la tupla recibida por la regla (`wTab`) y, a continuación, inicializa el atributo `id` con el valor devuelto por dicha consulta. El segundo *binding* ejecuta la consulta `wTab.@UserTable`, que obtiene todas las tuplas de la tabla `UserTable` que tienen




```
rule 'mapWebsite'
  from WebsiteTable wTab
  to   Website ws
  mapping
    ws.id    = wTab.id;
    ws.users = wTab.@UserTable;
end_rule
```

```
rule 'mapUser'
  from UserTable uTab
  to   User u
  mapping
    u.name = uTab.name;
end_rule
```

(b)

Fig. 1. Ejemplo simple de un proceso de transformación ScheMoL. (1) Entradas y salidas del proceso. (2) Definición de transformación utilizada en el ejemplo.

una clave ajena a la tupla recibida por la regla (`wTab`) y, a continuación, ejecuta aquella regla que conforme con el *binding*, que en este caso es la regla `mapUser`. Como en el ejemplo la consulta del segundo *binding* devuelve dos tuplas, la regla `mapUser` se ejecuta dos veces. La regla `mapUser` define la correspondencia entre la tabla `UserTable` y `User`. La ejecución de esta regla crea el elemento `User` e inicializa su único atributo (`name`) con el resultado de la consulta `uTab.name`, que devuelve el valor de la columna `name` para la tupla recibida por la regla.

References

1. Ulrich, W. M., Newcomb, P.H.: Information Systems Transformation: ADM Case Studies. Morgan Kaufmann, USA, (2010)
2. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: A model transformation tool. In Science of Computer Programming, 72(1-2), pp. 31-39 (2008).
3. Cuadrado, J. S., Molina, J.G., Tortosa, M. M.: RubyTL: A practical, extensible transformation language. In European Conference on Model Driven Architecture - Foundations and Applications, LNCS vol. 4066, pp. 158-172 (2006).
4. Mofscript project. http://www.eclipse.org/gmt/mofscript
5. Xpand website. http://wiki.eclipse.org/Xpand
6. Cánovas, J.L., García Molina, J.: A Domain Specific Language for Extracting Models in Software Modernization. In European Conference on Model Driven Architecture - Foundations and Applications, LNCS vol. 5562, pp. 82-97 (2009).
7. Reus, T., Geers, H., van Deursen, A.: Harvesting software systems for MDA-based reengineering. In European Conference on Model Driven Architecture - Foundations and Applications, LNCS vol. 4066, pp. 213-225 (2006).
8. Diaz, O., Puente, G., Cánovas, J.L., García Molina, J.: Harvesting models from web 2.0 databases. In Software and Systems Modeling, vol 11, pp. 1-20 (2011).