Gra2MoL: Una Herramienta para la Extracción de Modelos en Modernización de Software

Javier Luis Cánovas Izquierdo and Jesús García Molina

Grupo Modelum. Universidad de Murcia {jlcanovas,jmolina}@um.es

1 Introducción

La Modernización Dirigida por Modelos ha emergido recientemente como una nueva área dedicada a la automatización basada en modelos de procesos de modernización o evolución de software. Así, el OMG ha propuesto varios estándares de modernización dentro de la iniciativa ADM [1], como KDM [2]. En los próximos años será necesario un gran esfuerzo para encontrar técnicas y métodos para esta nueva área y será crucial disponer de herramientas que los soporten.

En general, el proceso de modernización asociado a un escenario de migración de plataformas consta de tres pasos principales: extracción de modelos a partir de los artefactos del sistema existente (ingeniería inversa), transformación de los modelos del sistema existente para generar modelos del sistema destino (rediseño) y generación de los artefactos del sistema destino (ingeniería directa). El paso dedicado a la extracción es realizado normalmente por soluciones *ad-hoc* basadas en la creación de analizadores específicos para realizar transformaciones texto-a-modelo. Con la finalidad de facilitar este paso, el grupo Modelum ha desarrollado un lenguaje de transformación texto-a-modelo denominado Gra2MoL [3, 4] que permite extraer modelos a partir de los ficheros de código fuente. De esta forma, una vez se dispone de los modelos del sistema, pueden ser llevadas a cabo las labores de reingeniería y generación del nuevo sistema aplicando transformaciones modelo-a-modelo o modelo-a-texto, completando el proceso de modernización.

2 Gra2MoL

Gra2MoL es un lenguaje basado en reglas que define un proceso de extracción de modelos como una transformación texto-a-modelo donde se especifican explícitamente las correspondencias entre los elementos de la gramática y los del metamodelo. La estructura de las reglas es similar a la considerada en lenguajes de transformación de modelos como ATL o RubyTL, con dos diferencias importantes: (1) el elemento origen de una regla es un elemento de la gramática en vez de un elemento del metamodelo y (2) la navegación es realizada por medio de un lenguaje de consultas adaptado para recorrer árboles de análisis, en vez de utilizar OCL o un lenguaje similar.

En Gra2MoL, cada regla especifica la correspondencia entre un elemento de la gramática y un elemento del metamodelo destino. Una regla tiene cuatro secciones: *from*, *to*, *queries* y *mappings*. La sección *from* especifica el símbolo

no terminal de la gramática origen y declara una variable que referenciará al nodo del árbol de análisis cuando la regla sea ejecutada. Esta variable puede ser utilizada por cualquier expresión dentro de la regla. Además, esta sección puede incluir opcionalmente un filtro con una condición para seleccionar los nodos que pueden ejecutar la regla. La sección *to* especifica la metaclase del elemento del metamodelo destino. La sección de *queries* contiene un conjunto de expresiones de consulta que permiten extraer información del árbol de análisis. El resultado de estas consultas puede ser utilizado en las asignaciones de la sección de *mappings*. Finalmente, la sección de *mappings* contiene un conjunto de *bindings* para asignar valores a las propiedades de los elementos del modelo destino en base a las variables definidas en la sección de consultas.

Gra2MoL incorpora un potente lenguaje de consultas inspirado en XPath que permite recorrer el árbol de análisis de forma eficiente y extraer información dispersa a lo largo del código. Una consulta consiste en una secuencia de operaciones que incluyen: un operador, un tipo de nodo y expresiones opcionales de acceso y de filtro. Hay tres tipos de operadores de consulta: `/`, `//` y `///`. El operador `/` devuelve los hijos inmediatos de un nodo y es similar al uso de la notación punto. Por otra parte, los operadores `//` y `///` permiten la navegación de todos los hijos del nodo (directos e indirectos) recuperando todos los nodos de un tipo dado. Estos dos operadores permiten ignorar nodos superfluos intermedios, facilitando la definición de la consulta, dado que se especifica qué tipo de nodos deben ser encontrados pero no cómo alcanzarlos. Dado que una consulta puede devolver uno o más subárboles, el operador `#` es utilizado en una de las operaciones de consulta para indicar el tipo de los nodos resultado. Las operaciones de consulta pueden incluir una expresión de filtro opcional tal que solo aquellos nodos que satisfagan dicha expresión serán seleccionados. También pueden incluir opcionalmente una expresión de acceso, la cual es utilizada para acceder a nodos hermanos del árbol de análisis por medio de índices.

La ejecución de una transformación Gra2MoL está dirigida por los *bindings* contenidos en la sección de *mappings*. Estos *bindings* son un tipo especial de asignación similar a la utilizada en lenguajes de transformación de modelos como RubyTL y ATL. La parte izquierda de la asignación debe ser una propiedad de la metaclase destino, mientras que la parte derecha puede ser un valor literal, un identificador de consulta o una expresión. En el primer caso, el valor se asigna directamente. En el segundo caso, la consulta es aplicada y se ejecuta aquella regla cuyos tipos de la sección *from* y *to* conforman con los tipos de la parte derecha e izquierda, respectivamente. Las reglas de conformancia son explicadas en [4]. Finalmente, en el tercer caso, la expresión es evaluada y si el resultado es un terminal, el valor se asigna directamente; si el resultado es un nodo, se ejecutará la regla asociada al *binding*.

3 Ejemplo

A continuación presentamos un ejemplo de Gra2MoL extraído de un proyecto de migración entre dos plataformas Java: Struts y JSF. Para favorecer la inter-

operabilidad se utilizó KDM como metamodelo para representar el código Java. Por cuestiones de espacio, en esta sección solamente se muestra un extracto de la definición de transformación Gra2MoL que extrae modelos a partir de código Java [1].

```
rule 'mapCodeModel'
  from compilationUnit cu
  to code::CodeModel
  queries
    clases : /cu/#normalClassDecl;
  mappings
    name = "codeModel";
    codeElement = classes;
end_rule

rule 'mapClassUnit'
  from normalClassDecl nc
  to code::ClassUnit
  queries
    elements : /nc/#classBodyDecl;
  mappings
    name = nc.Id;
    codeElement = elements;
end_rule
```

```
rule 'mapField'
  from classBodyDecl//fieldDecl
  to code::MemberUnit
  queries
    ...
  mappings
    ...
end_rule

rule 'mapMethod'
  from classBodyDecl//memDecl{Id.exists}
  to code::MethodUnit
  queries
    ...
  mappings
    ...
end_rule
```

Fig. 1. Fragmento de una transformación Gra2MoL para la extracción de modelos KDM a partir de código fuente Java.

La transformación arranca con la ejecución de la primera regla de la definición, llamada `mapCodeModel`, de modo que se crea una instancia de la metaclase `CodeModel`. Mientras que el primer *binding* de esta regla asigna un valor al atributo `name`, el segundo provoca la ejecución de la regla `mapClassUnit`, ya que conforma con las secciones *from* y *to* de dicha regla. La regla `mapClassUnit` tiene también un primer *binding* que asigna un valor al atributo `name` y el segundo binding provoca la ejecución de las reglas `mapField` y `mapMethod` dependiendo de los nodos resultado de la consulta `elements`, los cuales deben cumplir la condición del filtro *from* de estas reglas.

4 Descripción de la herramienta

La herramienta desarrollada para soportar Gra2MoL dispone de dos tareas Ant encargadas de realizar las labores principales de una transformación. Una de ellas se encarga de enriquecer la gramática del código fuente para que contemple la generación de un árbol de análisis a partir del código. A partir de la gramática enriquecida se obtiene un analizador con ANTLR. La segunda tarea se encarga de ejecutar la definición de la transformación Gra2MoL. Esta tarea tiene cuatro entradas: el analizador creado por la tarea anterior, el código fuente, la definición de la transformación y el metamodelo destino. Como resultado de esta tarea, se obtienen los modelos extraídos del código fuente.

Aunque la herramienta ha sido inicialmente diseñada para ser utilizada de forma totalmente independiente de la plataforma Eclipse, en la actualidad se

[1] La definición de transformación completa puede ser descargada de http://modelum.es/gra2mol

está llevando una labor de integración en la plataforma AGE [5]. AGE (*Agile Generative Environment*) es un entorno desarrollado también por el grupo Modelum cuyo principal objetivo es la experimentación con diferentes características de lenguajes de transformación, al mismo tiempo que es posible realizar desarrollos en contextos reales. Actualmente se dipone de un editor específico para las transformaciones Gra2MoL con resaltado de sintaxis, vista *outline* y autocompletado para la definición de las consultas, facilitando al desarrollador navegar por la gramática. La Figura 2 muestra una captura del editor y la vista *outline*.

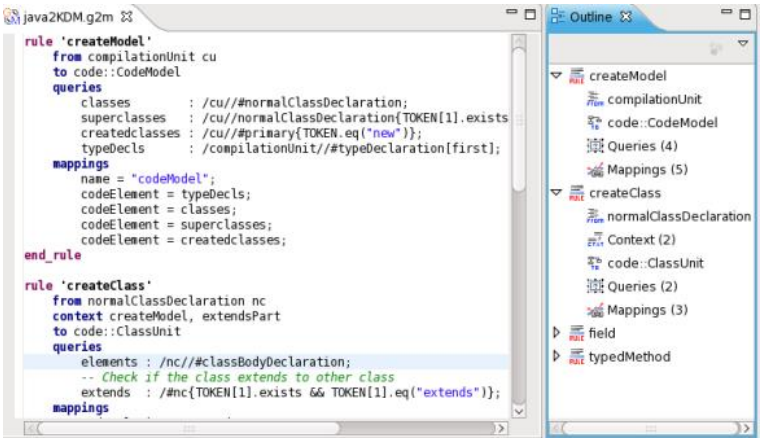

Fig. 2. Captura del entorno de desarrollo de la herramienta

Agradecimientos

Este trabajo ha sido parcialmente financiado por el proyecto 08797/PI/08 de la Fundación Séneca. Javier Luis Cánovas Izquierdo dispone de una beca FPI de la Fundación Séneca.

References

1. ADM. http://adm.omg.org
2. ADM Task Force: Knowledge discovery meta-model (kdm). OMG (2007).
3. J. Cánovas, O. Sánchez, J. S. Cuadrado, J. G. Molina, "DSLs para la extracción de modelos en modernización". V Taller sobre Desarrollo de Software Dirigido por Modelos enmarcado en las XIII Jornadas de Ingeniera del Software y Bases de Datos (2008).
4. J. Cánovas, J. G. Molina, "A domain specific language for extracting models in software modernization". V European Conference on Model-Driven Architecture Foundations and Applications, ECMDA09 (2009).
5. J. S. Cuadrado, J. G. Molina, "AGE (Agile Generative Environment)". Sesión de demostraciones de herramientas en XII Jornadas de Ingeniería del Software y Bases de Datos (2007).