Análisis de transformaciones de modelos ATL con AnATLyzer

Jesús Sánchez Cuadrado, Esther Guerra and Juan de Lara

Grupo de investigación Miso (`http://miso.es`)
Universidad Autónoma de Madrid (Spain)

Resumen Las transformaciones de modelos son un elemento clave del Desarrollo de Software Dirigido por Modelos puesto que permiten automatizar muchas tareas de manipulación de modelos. Por tanto, disponer de métodos que permitan detectar errores no triviales resulta esencial. Sin embargo no existen herramientas prácticas de análisis de transformaciones que sean capaces de tratar con transformaciones complejas.
En esta demostración se presentará anATLyzer, un analizador estático para transformaciones ATL que hace uso de *constraint solving* para mejorar la precisión del análisis. anATLyzer no se limita a un subconjunto de ATL sino que intenta cubrir ATL completamente. Se integra con el editor de ATL en Eclipse, y ofrece servicios adicionales como visualización y quick fixes, así como una API para ser utilizado de manera programática.
La demostración se ilustrará con un ejemplo sobre el que se mostrarán algunos de los tipos de errores que hemos encontrado analizando transformaciones del Zoo de ATL, con el objetivo de motivar la necesidad de este tipo de herramientas y mostrar sus características principales.

Keywords: Transformaciones de modelos, ATL, Análisis estático, Desarrollo dirigido por modelos.

1. Introducción

Las transformaciones de modelos son un elemento clave del Desarrollo de Software Dirigido por Modelos (DSDM), puesto que permiten automatizar la manipulación de los modelos. Aunque en los últimos años se han propuesto toda una serie de lenguajes de transformación [3,4,5,7], las implementaciones de estos lenguajes han prestado poca atención al análisis de las transformaciones escritas con ellos. Así, la situación actual es que muchos lenguajes populares como ATL [4] y ETL [5] no tienen ningún tipo de análisis estático previo a la ejecución, lo que lleva a costosos ciclos de ejecución-prueba para depurar la transformación. Otros lenguajes, como QVT [7], disponen de cierto chequeo de tipos pero no ofrecen ningún tipo de análisis relativo a la ejecución de reglas.

Para abordar esta problemática hemos creado anATLyzer, un analizador estático de transformaciones ATL. La siguiente sección presenta sus características principales, y la Sección 3 describe brevemente el contenido de la demostracioń que se realizará.

2. AnATLyzer

El objetivo de anATLyzer es la detección avanzada de errores en transformaciones ATL [1]. El proceso de análisis tiene tres partes. Primero se lleva a cabo un chequeo de tipos, en el que se realiza la inferencia de tipos de las expresiones OCL y se asigna a cada nodo del árbol sintáctico un tipo. Esta etapa no es trivial ya que ATL está tipado dinámicamente y su compilador no emite errores más allá de cuestiones sintácticas. En la segunda etapa se analizan las relaciones entre reglas y se detectan problemas asociados. En estas dos primeras etapas, con el objetivo de mejorar la precisión del análisis, algunos errores se marcan como potenciales. En la tercera etapa del análisis se utiliza satisfacción de restricciones (*constraint solving*) para confirmar o descartar estos errores potenciales. Para cada error potencial se calcula su *path condition* como una expresión OCL que describe las características de aquellos modelos que harían que el flujo de la transformación llegara a la localización que contiene el problema. Esta expresión se utiliza como entrada de la herrramienta USE Validator [6], que intenta encontrar un modelo que la cumpla. Si tal modelo existe significa que el error es real ya que existe al menos una configuración que haría fallar la transformación, mientras que en caso contrario se descarta el error potencial.

Nuestra técnica de análisis amplía el rango de errores que se pueden detectar y permite un uso eficiente del *constraint solver* mediante la poda de los meta-modelos de entrada (*pruning*). Como dato significativo, en nuestros experimentos con transformaciones del Zoo de ATL[1] el tiempo de ejecución del *constraint solver* es en la mayoría de los casos inferior a medio segundo. También cabe destacar que todas las transformaciones del zoo tienen al menos un problema. La Tabla 1 muestra algunos de los errores más significativos que se detectan con anATLyzer. El analizador actualmente detecta 45 tipos distintos de error.

	Descripción	**Tipo de error**	**Precisión**
1	Propiedad u operación no encontrada	tipado	estática
2	Declaración no coincide con el tipo inferido	tipado	estática
3	Acceso a objeto sin definir (OclUndefined)	navegación	solver
4	Propiedad obligatoria no inicializada	integridad del modelo destino	estática
5	Binding resuelto por regla con tipo incompatible	integridad del modelo destino	solver
6	Binding no resuelto	reglas	solver
7	Conflicto de reglas	reglas	solver

Tabla 1: Algunos de los problemas detectados por anATLyzer.

Entre los errores que anATLyzer detecta se incluyen el acceso a una propiedad no definida en el meta-modelo (error #1) o si los tipos declarados en la transformación no coinciden con el tipo inferido (error #2). Este tipo de problemas están relacionados con el tipado de la transformación, no requieren el uso del *constraint solver*, y decimos que su precisión es "estática". También se detectan errores relativos a la navegación en expresiones OCL, como por ejemplo el acceso a un objeto no definido (i.e., un "null pointer exception", error #3).

[1] `http://www.eclipse.org/atl/atlTransformations/`

Este tipo de error a veces requiere confirmación mediante el uso del *constraint solver*. Una clase importante de errores son aquellos que afectan a la integridad del modelo generado, esto es, si la transformación genera siempre modelos que son conformes al meta-modelo destino. Por ejemplo, es común olvidar la inicialización de una propiedad obligatoria al crear un objeto destino (error #4). Un error más sutil, que requiere *constraint solving* para su detección, es la asignación incorrecta en una propiedad debido a que la regla que resuelve el *binding* correspondiente tiene un tipo incorrecto en su parte *to* (error #5). También se detectan situaciones en las que puede que un *binding* no sea resuelto por ninguna regla (error #6) o conflictos de reglas (i.e., reglas cuyos patrones de entrada no son exclusivos, error #7).

De cara al desarrollador de transformaciones, ANATLYZER se integra con el editor estándar de ATL en Eclipse para realizar el análisis estático al mismo tiempo que se desarrolla la transformación. La Figura 1 muestra una captura de pantalla del entorno. La etiqueta 1 muestra el editor de ATL con marcadores de error producidos por ANATLYZER. Estos marcadores también aparecen en la vista de problemas de Eclipse, y además, se ha creado una vista específica para el análisis de transformaciones ATL (etiqueta 2). La ejecución del *constraint solver* se realiza en segundo plano y se ha implementado un mecanismo para que su ejecución sea incremental, es decir, cuando se cambia la transformación sólo hay que volver a invocarlo para aquellos errores afectados por el cambio. Por otra parte, dado un error es interesante disponer de mecanismos automatizados para arreglarlo. Para esto se ha implementado un amplio catálogo de *quick fixes* [2] (etiqueta 3). Finalmente, se está trabajando en la visualización de los problemas para facilitar su comprensión (etiqueta 4).

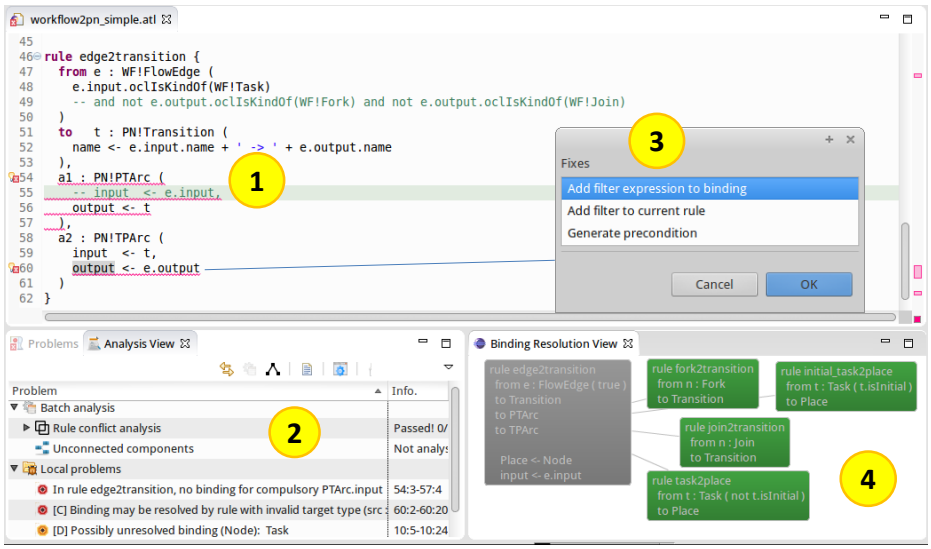

Figura 1: Entorno de desarrollo en Eclipse

Por último, anATLyzer ofrece una API que permite invocar al analizador de manera programática, inspeccionar los resultados, y obtener una versión del árbol sintáctico de ATL enriquecida con información de tipos. Creemos que esta característica es muy útil para quienes investigan en transformaciones ATL.

3. Demostración

Esta demostración pretende dar a conocer anATLyzer a la comunidad española de DSDM con tres objetivos principales. Por una parte, animar a los programadores ATL a que prueben la herramienta y tengan la oportunidad de descubrir si sus transformaciones tienen errores. También, como se ha mencionado, pensamos que anATLyzer es de utilidad para cualquier investigador interesado en el análisis de transformaciones. Por último, estamos interesados en obtener realimentación acerca de *bugs*, mejoras y sobre la utilidad de la herramienta por parte de que aquellos que la usen.

En la demostración se presentarán las características de anATLyzer, utilizando como guía ejemplos simples pero inspirados en casos reales encontrados en transformaciones del Zoo de ATL. A través de estos ejemplos se mostrarán los errores más importantes que se pueden detectar y se motivará la utilidad de la herramienta. También se mostrarán otros aspectos como su configuración, rendimiento, *quick fixes*, etc. La herramienta está disponible en `http://miso.es/tools/anATLyzer.html`, donde se incluye el código fuente, un *update site* para su instalación, *screencasts* y documentación.

Agradecimientos. Trabajo financiado por el MINECO (TIN2014-52129-R), la Comunidad de Madrid (S2013/ICE-3006) y la Comisión Europea (FP7-ICT-2013-10, #611125).

Referencias

1. J. S. Cuadrado, E. Guerra, and J. de Lara. Uncovering errors in atl model transformations using static analysis and constraint solving. In *ISSRE'14*, pages 1–11. IEEE Computer Society, 2014.
2. J. S. Cuadrado, E. Guerra, and J. de Lara. Quick fixing ATL model transformations. In *MoDELS'15*, pages 146–155. IEEE, 2015.
3. J. S. Cuadrado, J. G. Molina, and M. M. Tortosa. RubyTL: A Practical, Extensible Transformation Language. In *ECMDA-FA'06*, volume 4066 of *LNCS*, pages 158–172. Springer, 2006.
4. F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. ATL: A model transformation tool. *Science of Computer Programming*, 72(1-2):31 – 39, 2008.
5. D. S. Kolovos, R. F. Paige, and F. Polack. The epsilon transformation language. In *ICMT'08*, volume 5063 of *LNCS*, pages 46–60. Springer, 2008.
6. M. Kuhlmann, L. Hamann, and M. Gogolla. Extensive validation of OCL models by integrating SAT solving into USE. In *TOOLS (49)*, volume 6705 of *LNCS*, pages 290–306. Springer, 2011.
7. QVT. `http://www.omg.org/spec/QVT/`.