Integrando Modelos en UML y Especificaciones Formales

Ana Funes, Arístides Dasso, Daniel Riesco, Germán Montejano

Universidad Nacional de San Luis
Ejército de los Andes 950
5700 – San Luis (Argentina)
Tel: +54 – 2652 – 42 – 4027 int. 251
Fax: +54 – 2652 – 43 – 0059
{afunes, arisdas, driesco, gmonte}@unsl.edu.ar

Resumen En el presente informe se dan los lineamientos generales de un proyecto de desarrollo de software para la integración de una herramienta gráfica de diseño de diagramas de clases en UML y un generador de especificaciones formales en RSL. Esta integración permitirá que un ingeniero de software realice especificaciones formales asociadas a modelos semi-formales en UML trabajando en un ambiente integrado de desarrollo.

1 Introducción

El Lenguaje Unificado de Modelado (UML) [1], [17], [19] es un lenguaje gráfico para modelar y especificar sistemas de software. Hoy en día se ha transformado de hecho en el lenguaje estándar de modelado orientado a objetos. Sin embargo, aunque las notaciones en UML son fácilmente comunicadas al usuario final, su semántica es informal y, en consecuencia, puede resultar ambigua.
En la literatura se pueden encontrar varios trabajos que tratan sobre la formalización de la semántica y la sintaxis de distintas partes de UML (ver [2], [4], [5], [6], [7] [12], [13], [16]).
En [3], [8], [15] y [18] se hace uso de RSL (the RAISE Specification Language) como base formal para los diagramas de clases de UML. RSL es un lenguaje de especificación formal [9], el cual recibe su nombre del método RAISE. RAISE (Rigorous Approach to Industrial Software Engineering) [10] consiste de un número de técnicas y estrategias para realizar desarrollo formal. Su lenguaje, RSL, es un lenguaje de especificación de amplio espectro. Permite el uso de diferentes estilos de especificaciones: aplicativo o imperativo; secuencial o concurrente; directo (explícito) o axiomático (implícito); algebraico (con tipos de datos abstractos) u orientado al modelo (con tipos de datos concretos).
En este trabajo presentamos los puntos principales de un proyecto de desarrollo de software para integrar una herramienta gráfica para la construcción de diagramas de clases en UML con un generador de especificaciones formales en RSL basado en la semántica y sintaxis presentadas en [8]. Esta nueva herramienta integrada de desarrollo permitirá al Ingeniero de Software construir sus propios diagramas de clases y generar a partir de éstos las correspondientes especificaciones formales en RSL asociadas a los mismos.
Una de las premisas del proyecto está orientada a favorecer el desarrollo y empleo de software de libre distribución.

2 Antecedentes

El proyecto cuenta ya con una herramienta de desarrollo propio, UML2RSL, la cual permite la generación automática de especificaciones formales en RSL a partir de diagramas de clases en UML.

Dicha herramienta toma como entrada un archivo XML, el cual contiene una descripción de un diagrama de clases, lo analiza sintácticamente y produce como salida una especificación formal en RSL.
Sin embargo, dado que para generar este tipo de archivos XML, usados como entrada por el traductor, de momento es necesario hacer uso de una herramienta comercial, se ha empezado a trabajar en la integración de nuestro traductor con una herramienta gráfica de libre distribución basada en UML. En este sentido, se ha optado por llevar a cabo dicha integración con Argo UML.

2.1 UML2RSL

Como se muestra en la figura 1, UML2RSL usa como entrada un diagrama de clases producido por una herramienta comercial, el cual se encuentra almacenado en un archivo en formato XMI, un formato basado en XML [11], [14] para intercambio de información entre herramientas de UML.
Analiza sintácticamente el archivo XML de entrada, y si la entrada es válida, produce una especificación RSL basada en las semánticas propuestas en [8].
Esta herramienta fue desarrollada en Java y corre en cualquier plataforma Java 2. Hace uso de una API muy utilizada para procesar archivos XML: la API DOM (Document Object Model) [11], [14].
Haciendo uso de la API DOM, se analiza el documento XML para verificar que es un documento XML válido, en cuyo caso es representado como un árbol DOM. DOM nos provee un conjunto de APIs para manipular los nodos de un árbol DOM. Haciendo uso de estas APIs se analiza si el árbol corresponde a la descripción de un diagrama de clases de acuerdo al DTD XMI y no a otro documento XML, además de controlar que el diagrama de clases sea sintácticamente correcto de acuerdo a la sintaxis formal presentada en [8]. A medida que se lleva a cabo este análisis se va creando una nueva estructura dinámica en memoria que contiene toda la información del diagrama relevante para la traducción.

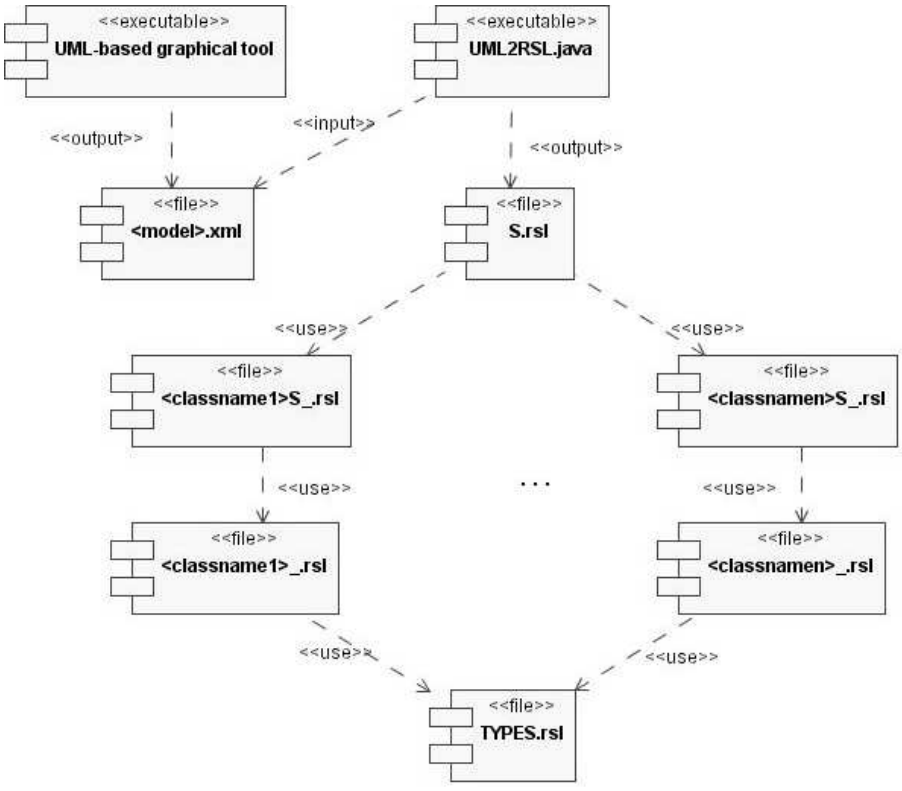

Figura 1. Diagrama de componentes

2.2 Argo UML

Argo UML es una herramienta desarrollada en Java para el diseño de modelos en UML. Permite crear la mayoría de los diagramas estándares de UML, en particular los diagramas de clases que son los que a nosotros nos interesan en este caso.
Argo UML se encuentra disponible bajo una licencia de código abierto y cuenta con la capacidad de soportar la generación de archivos XMI. Usa este mecanismo estándar de grabación de los modelos creados, para facilitar el intercambio con otras herramientas.
Una descripción mas completa sobre la herramienta y el proyecto pude obtenerse en http://argouml.tigris.org

3 La integración de ambas herramientas

La integración puede ser llevada a cabo en diferentes formas:

- Dado que existen una variedad de formatos XMI generados por distintas aplicaciones, en particular este es el caso entre el formato de los archivos XMI de salida producidos por Argo UML y el formato de los archivos XMI de entrada aceptados por UML2RSL, la integración podría llevarse a cabo agregándole un módulo al traductor de manera tal que sirva de puente entre la salida XMI producida por Argo UML y el tipo de formato XMI reconocido por UML2RSL.
- Crear una nueva versión del traductor compatible con Argo UML, en donde se cambie el módulo correspondiente al análisis sintáctico del archivo XMI de entrada, de manera tal que ahora pueda reconocer los archivos XMI compatibles con Argo UML.
- Dado que Argo UML es una herramienta “Open Source”, teniendo acceso a su código fuente se puede trabajar en la integración dentro del código de Argo UML del módulo del traductor que es encargado de producir las especificaciones formales en RSL. En este caso, no es necesario trabajar con los archivos XMI sino con las estructuras generadas internamente por Argo UML y aquella que usa el traductor para generar las especificaciones.

Referencias

1. G. Booch, J. Rumbaugh, and I. Jacobson. The Unified Modeling Language User Guide. Addison-Wesley, 1999.
2. R. Breu et al., Towards a formalization of the Unified Modeling Language. In Proceedings of ECOOP’97, number 1241 in LNCS, Springer-Verlag, 1997.
3. N. Debnath, D. Riesco et al., “Using UML class diagram for RAISE Applicative Specification”, ACIS International Journal of “Computer & Information Science”, Vol. 3, Pages 84-93, March 2002. ISBN/ISSN 1525-9293.
4. France,R., Lano,K. and Rumpe, B., Developing the UML as a formal modeling notation, UML’98 Beyond the notation, Muller and Bezivin editors, number 1618 in LNCS, Springer-Verlag, 1998.
5. A. Evans et al., Towards a core metamodelling semantics of UML, Behavioral Specifications of Businesses and Systems, H. Kilov editro, Kluwer Academic Publishers, 1999.
6. R. France. A Problem-Oriented Analysis of Basic UML Static Requirements Modeling Concepts. In Proceedings of OOPSLA '99, Denver, CO, USA, Nov. 1999.

7. R. France and B. Rumpe, editors. Proceedings of the UML´99 conference, Beyond the Standard, Colorado, USA, number 1723 in LNCS, Springer-Verlag, 1999.
8. A. Funes and C. George. , Chapter 8: "Formalizing UML class diagrams" in "UML and the Unified Process", ISBN 1931777446, Idea Group Publishing; April 1, 2003.
9. C. George et al. The RAISE Specification Language. Prentice-Hall International (UK) Limited, 1992.
10. C. George et al. The RAISE Development Method. Prentice-Hall International (UK) Limited, 1995.
11. Holzner, S. Inside XML. New Riders, 2001.
12. S. Kim and D. Carrington, A Formal Specification Mapping between UML Models and Object-Z Specifications. In LNCS, number 1878, pages 2–21. Springer-Verlag, 2000.
13. K. Lano and J. Biccaregui, Formalizing the UML in Structured Temporal Theories, 2nd ECOOP Workshop on Precise Behavioral Semantics, TUM-I9813, Technische Universitat Munchen, 1998.
14. Maruyama, H., Tamura, K., and Uramoto, N. XML and Java, Developing Web Applications. Addison-Wesley, 2000.
15. G. Montejano et al., "RAISE Formalization of UML Class Associations", In Proceedings of CAINE 2000, Honolulu, Hawai, USA, Nov 1-3, 2000.
16. P. Muller and J. Bezivin, editors. Proceedings of the UML´98 conference, Beyond the notation, Mulhouse, France, number 1618 in LNCS , Springer-Verlag, 1998.
17. OMG. OMG-Unified Modeling Language v1.4, chapter UML Notation Guide. http://www.omg.org/technology/documents/formal/uml.htm, September 2001.
18. D. Riesco et al., "UML Class Structure Interpretation using RAISE Abstract Applicative Specification", In Proceedings of CAINE 2000, Honolulu, Hawai, USA, Nov 1-3, 2000.
19. J. Rumbaugh, I. Jacobson, and G. Booch. The Unified Modeling Languge Reference Manual. Object Technology Series. Addison Wesley, 1999.