Darwin, Una Herramienta para Verificar Diseños Arquitecturales

Pablo Castro Pablo Ponzio
Ramiro Demasi
Universidad Nacional de Río Cuarto
Departamento de Computación
e-mail: {pcastro,pponzio,rdemasi}@dc.exa.unrc.edu.ar
Gabriel Baum
LIFIA - Universidad Nacional de La Plata
e-mail: gbaum@sol.info.unlp.edu.ar

cmr

Resumen

En este trabajo presentamos una herramienta que permite a los ingenieros de software realizar ciertas verificaciones sobre diseños arquitecturales. Esta herramienta, llamada Darwin, utiliza como lenguaje de modelado la notación BON, en cuanto las verificaciones son realizadas por medio del lenguaje lógico Alloy. Adicionalmente presentamos algunas de las funcionalidades que agregaremos a la herramienta en un futuro próximo.

1. Introducción

Con la aparición del paradigma de orientación a objetos, surgieron diferentes lenguajes de modelado para soportar esta nueva tecnología. En los últimos tiempos apareció como una alternativa a los lenguajes más reconocidos (por ejemplo, UML [9]) el lenguaje de modelado BON (Bussiness Object Notation [3]) En [4] se muestra como este lenguaje permite una aplicación transparente de los métodos formales. Esto se debe principalmente a su notación simple, y a que fue diseñado para soportar el método de Diseño por Contratos, incluyendo un lenguaje formal de aserciones. Otras características importantes de BON son *Seamlessness* (desarrollo sin transiciones), que intuitivamente significa que el paso de una etapa en el

desarrollo a la siguiente es directo, y Reversibilidad, que dice que los cambios realizados en una etapa del desarrollo son reflejados en las anteriores. Estas favorecen el reuso y simplifican el mantenimiento del software.

El presente trabajo se basa en la integración de dos herramientas que pueden ser utilizadas en la etapa de diseño del desarrollo de un sistema de software, la notación BON, y el lenguaje formal Alloy. Basándonos en [4], implementamos una herramienta, que llamamos *Darwin Tool*, la cual permite editar diagramas BON y realizar automáticamente la traducción del modelo al lenguaje lógico Alloy, lo que permite verificar propiedades del modelo utilizando la herramienta Alloy Analyzer. Otra motivación para la realización de una herramienta CASE para BON es que actualmente existe una carencia de herramientas que soporten esta notación. La más importante de ellas es EiffelCase de ISE, la cual es privativa [1].

El presente artículo está estructurado de la siguiente forma. En la sección 2 se presenta el diseño de la herramienta. En la sección 3 mostramos un ejemplo de aplicacion de Darwin. Finalmente, presentamos las conclusiones y los trabajos futuros.

2. Diseño de Darwin

Darwin está dividida en dos modulos principales. Por un lado, posee una interfaz gráfica para la interacción con el usuario, que tiene las características clásicas de una herramienta CASE. Por otra parte, se encuentra un parser traductor llamado Bon2Alloy el cual permite traducir modelos descritos en la notación textual de BON a especificaciones Alloy. Más detalladamente:

- Darwin Tool: Parte gráfica de la herramienta, la cual fue construída utilizando las librerías wxWidgets y OGL (ver [8]). Esta herramienta, además de las funcionalidades clásicas, permite generar código textual BON a partir de un modelo gráfico.

- Bon2Alloy: Parser traductor que toma como entrada un modelo en notación textual BON y genera una especificación Alloy correspondiente al modelo dado. Podemos decir que esta aplicación posee dos partes, por un lado un parser que construye un conjunto de objetos que son una abstracción de un modelo BON. Los cuales, además poseen las funcionalidades necesarias para producir el código Alloy correspondiente.

La figura 1 muestra una intuición sobre la forma en que funciona Darwin. A continuación daremos una explicación detallada del diagrama:

[1] Darwin Tool ha sido desarrollada bajo el paradigma GNU.

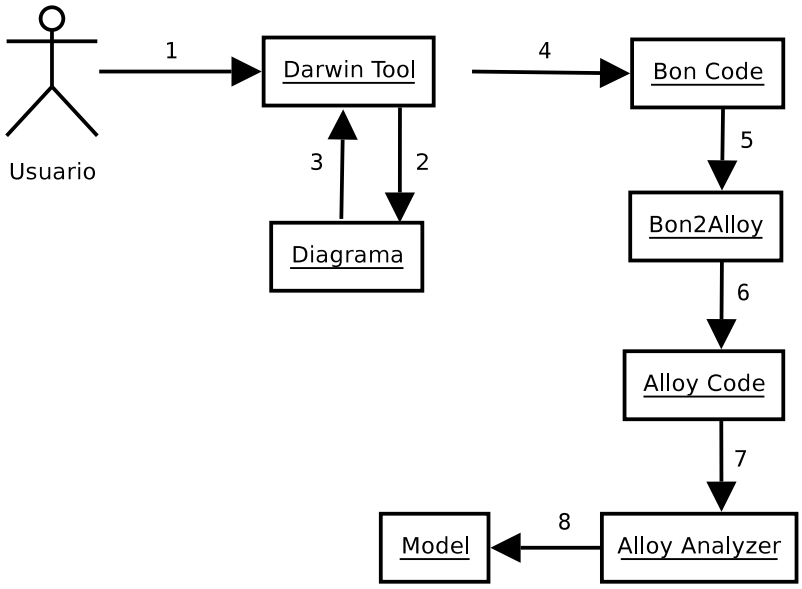

Figura 1: Interacción entre los Componentes de Darwin

1. El usuario edita un diagrama BON en Darwin Tool (1, 2 y 3).
2. Darwin Tool genera el código BON asociado al diagrama (4).
3. Luego realiza una llamada a Bon2Alloy, la cual automáticamente traduce el código BON a una especificación Alloy (5 y 6).
4. Finalmente se realiza una llamada al Alloy Analyzer, con la especificación Alloy como entrada, y esta genera un modelo (cuando es posible) que satisface la especificación dada (7 y 8).

Es importante resaltar la flexibilidad que provee la separación en módulos, ya que es posible desarrollar ambas partes en paralelo sin afectar la integridad del programa.

3. Un Caso de Estudio

Hemos testeado el funcionamiento de la herramienta con varios casos de estudio, en este artículo mostramos sólo uno de ellos por cuestiones de espacio. El siguiente gráfico presenta en detalle un diseño BON.

Este diseño fue traducido automáticamente por Darwin y verificado por la herramienta Alloy, obteniendo un modelo del diseño que asegura su consistencia.

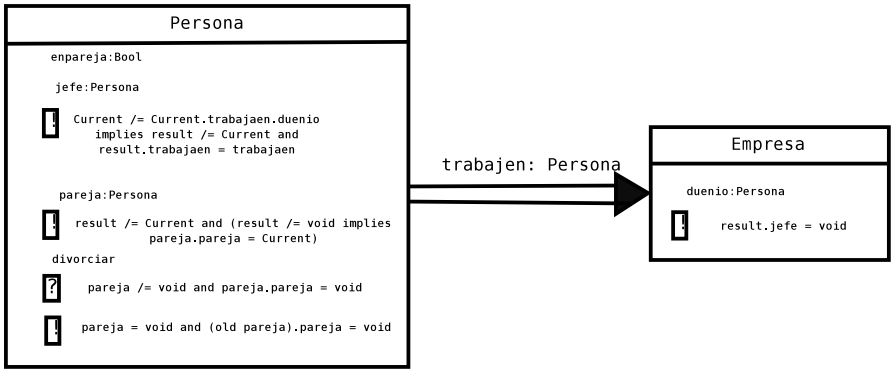

Figura 2: Caso de Estudio

4. Conclusiones

Consideramos que este proyecto forma parte de los primeros pasos en la integración de métodos formales con las técnicas de diseño informales, las cuales son usadas ampliamente en la actualidad. La poca utilización de los métodos formales en la industria se debe a que requiere cierta clase de conocimientos matemáticos por parte del usuario, y este tipo de entrenamiento es costoso en tiempo y dinero. Pero también sabemos que, en aplicaciones críticas, los métodos formales son necesarios si se quiere asegurar el correcto funcionamiento de las mismas. Aquí es donde Darwin Tool entra en escena, ya que, permite utilizar métodos formales para verificar algunas propiedades de un diseño realizado en la notación BON, esto implica que la utilización de formalismos es intuitiva para el usuario, es decir, no necesita tener ningún tipo de conocimientos adicionales para aplicarlos. Este enfoque es completamente diferente al de las herramientas existentes en la actualidad, como por ejemplo, *z-eves* que permite modelar en el lenguaje Z, Isabelle/HOL, el demostrador semi-automático de teoremas, e inclusive el mismo Alloy, ya que, fueron pensadas para usuarios con una preparación matemática adecuada.

5. Trabajos Futuros

- Soportar la forma expandida de las clases BON y los diagramas de secuencia de la misma notación.
- Realizar chequeos sobre las aserciones introducidas por el usuario, como por ejemplo, que sean correctas en cuanto a tipos.
- Permitir mostrar al usuario el resultado del chequeo (mediante Alloy Analyzer) del modelo generado por Bon2Alloy, de forma tal que no requiera

conocimientos sobre Alloy para poder interpretar los resultados, logrando la integración completa del método BON con Alloy.

- Extender el lenguaje BON para soportar operadores de la lógica relacional, como por ejemplo, la clausura transitiva.
- Soportar los cuantificadores existencial y universal en las aserciones BON.
- Considerar la traducción de features BON que incluyan parámetros.
- Permitir el chequeo de herencia y de invariantes de clase, lo que permitiría chequear el cumplimiento del principio de subcontratación.
- Realizar chequeos de sintaxis y tipos antes de realizar la traducción, para mejorar la eficiencia, ya que, estos errores actualmente son manejados por el Alloy Analyzer.

Referencias

[1] D. Jackson. Alloy 3.0 Reference Manual. May 2004.

[2] D. Jackson. Micromodels of Software: Lightweight Modeling and Analisis with Alloy. February 2002.

[3] Kim Waldén and Jean-Marc Nerson. Seamless Object-Oriented Software Architecture.

[4] Castro Pablo, Baum Grabriel. Integrando BON con Alloy**.**

[5] Castro Pablo, Baum Gabriel. Utilizando Contratos de Reuso con Alloy. *Anales del CASIC 2001, 2001.*

[6] Castro Pablo, Baum Gabriel. Maquinas Evolutivas. *Anales del CASIC 2002, 2002.*

[7] Castro Pablo, Baum Gabriel. Una Formalización del Proceso de Desarrollo. *Anales del CLEI 2003, 2003.*

[8] Julian Smart, Robert Roebling, Vadim Zeitlin, Robin Dunn. wxWindows 2.4.2: A portable C++ and Python GUI toolkit. September 2003.

[9] J.Rumbaugh G.Booch and I.Jacobson. The Unified Modeling Language User Guide. Addison-Wesley, 1999.

[10] Richard F. Paige and Jonathan S. Ostroff. A Comparation of Business Object Language and the Unified Modeling Language.