Programación Generativa en Java y Modificación de Funcionalidades a Tiempo de Carga

José Ismael Beristain Colorado, Ulises Juárez Martínez, Luis Reyes Hernández

Instituto tecnológico de Orizaba, División de estudios de posgrado e investigación.
Oriente 9 No. 854, Col. Emiliano Zapata, Orizaba, Ver., México, 94320.
isma.sp6@gmail.com, ujuarez@ito-depi.edu.mx, l.a.reyes.h@gmail.com

2014 Published by $\mathcal{DIFU}_{100}ci@$ http://nautilus.uaz.edu.mx/difu100cia

Resumen

La programación generativa (PG) es un paradigma de desarrollo de software que modela e implementa familias de sistemas, permitiendo a un sistema generarse automáticamente con base en una especificación definida, teniendo como objetivo conseguir alta intención, reutilización y adaptación sin comprometer el desempeño en tiempo de ejecución ni los recursos del software, solventando la necesidad de adaptación de una aplicación a nuevos requerimientos. En este artículo se presenta la situación actual de la PG y se describe el caso de estudio de una aplicación para química, representando la creación de elementos químicos por medio de la creación de objetos; sin embargo, en ocasiones es necesario tener una versión extendida de dicha aplicación, por tanto se utilizan herramientas de meta-programación y el mecanismo de entrelazado de aspectos a tiempo de carga para agregar la funcionalidad de representar la creación de moléculas con base en los elementos creados por el sistema original.

Palabras clave: Java, Load-Time Weaving, Programación generativa.

1. Introducción

Actualmente la etapa de mantenimiento presenta un continuo cambio en los requerimientos y la necesidad de contar con un buen manejo de versiones del sistema; la programación orientada a objetos (POO) y programación orientada a aspectos (POA) proporcionan cierto nivel de encapsulación y modificación de programas facilitando el reemplazo y mantenimiento de software, sin embargo no permiten realizar modificaciones a un sistema de manera automatizada y a tiempo de ejecución. La PG brinda el soporte adecuado para solventar esta necesidad. La PG es un paradigma que permite el análisis, desarrollo y mantenimiento de aplicaciones de software, permitiendo la generación de programas con características muy similares a través de líneas de productos de software (LPS), teniendo tanta similitud es posible el intercambio de las piezas de software con que están construidos dichos programas; el análisis de las LPS brinda un amplio conocimiento

del comportamiento de los sistemas, permitiendo prever cuáles piezas tienen más tendencia a ser modificadas y sus posibles cambios. También es posible reutilizar elementos previamente construidos generando nuevos programas de manera automatizada, siendo necesario el soporte de distintas técnicas de programación y meta-programación. Las bases de la PG son:

- Programación intencional.- Es un paradigma cuyo objetivo es que la implementación de un sistema refleje el comportamiento específico (intención) que el programador tiene en mente.
- POA.- Es un paradigma de programación que permite una mejor modularización del sistema a través de la separación de asuntos, entrelazando (a tiempo de carga, compilación o ejecución) los módulos que sean necesarios para el correcto funcionamiento de un programa.
- Generadores.- Un generador es un componente que obtiene como entrada una especificación (intención) de un requerimiento nuevo o cambiante y mediante técnicas de meta-programación realiza de manera automática los cambios necesarios o genera nuevo código que satisfaga el cambio de requerimientos.

La programación generativa ha recibido poca atención por la industria de desarrollo de software en su aplicación práctica, siendo poco común que en términos de desarrollo se contemple un enfoque de PG para la solución de problemas. Esto debido a la falta de conocimiento en tecnologías propias del lenguaje y del desarrollo de aplicaciones que abarcan líneas de productos de software. La contribución principal de este trabajo consiste en ilustrar mediante un caso de estudio, las ventajas de implementar mecanismos de meta-programación para generar y transformar programas en tiempo de carga bajo el enfoque Java. Este artículo se organiza de la siguiente forma: La sección 2 describe la situación actual de la PG desde un enfoque general. La sección 3 se refiere a la PG enfocada a la tecnología Java. La sección 4 describe el caso de estudio y muestra la implementación propuesta. La sección 5 muestra los resultados obtenidos. La sección 6 da pie a la discusión de ideas. La sección 7 presenta las conclusiones y trabajo a futuro.

2. Situación actual de la PG

La aplicación de la PG presenta un enfoque de auto-configuración y generación automática de componentes: en [1] se presenta un lenguaje específico del dominio para generar código que permita la evolución acoplada del modelo relacional de una aplicación con su modelo de datos sin romper las abstracciones provistas por el sistema. En [2] se describe a GeoGram como un sistema que genera programas para cómputo geométrico, mediante la combinación de componentes de software genéricos realiza inferencias para derivar nuevos datos, introducir nuevos objetos, filtrar las opciones presentadas al usuario y generar el programa. En [3] se presenta un framework para monitorizar el código generado en tiempo de ejecución aumentando el nivel de abstracción con el que los desarrolladores analizan el código obtenido. En [4] se propone introducir la PG en el área de generación de aplicaciones de interfaces de usuario gráficas para generar aplicaciones personalizadas de forma automática mediante especificaciones abstractas. En [5] se mencionan los generadores de bibliotecas de alto rendimiento, comparando las características y conceptos necesarios para que un lenguaje de meta-programación permita la construcción sistemática de estos. En [6] se propone un proceso para el manejo de cambios a nivel de features en el desarrollo de software para solventar problemas de ineficiencia en la comunicación, fallas en el código y altos costos de mantenimiento que se presentan en los proyectos de software debido a la diversidad de stakeholders y artefactos.

3. Programación generativa en Java

En el ambiente Java también se reportan trabajos relacionados con la programación generativa, por ejemplo: en [7] se modificó una Máquina Virtual de alto rendimiento para realizar cambios a clases cargadas, permitiendo modificarlas en cualquier momento durante la ejecución del programa. En [8] se utiliza la generación automática de código para migrar bibliotecas de una Máquina Virtual de Java estándar a una empotrada. En [9] se presenta una herramienta basada en Java HotSpot la cual permite realizar cambios en tiempo de ejecución a las clases cargadas, basándose en esta herramienta se desarrolló una versión mejorada de una parte del IDE NetBeans, añadiendo componentes sin reiniciar la aplicación. En [10] se desarrolló una biblioteca de clases que genera el código necesario para que los programas escritos por los desarrolladores cumplan con los protocolos necesarios que establece cada framework que se necesita implementar, reduciendo costos al ahorrar el tiempo que invierte el desarrollador para aprender a utilizar cada framework. Algunas herramientas de meta-programación que se encuentran bajo el enfoque Java: Load Time Weaving (LTW) permite el entrelazado de

aspectos a tiempo de carga, promoviendo la mínima invasión al código original, para esto es necesario un agente que se encarga de interceptar las clases que deben ser entrelazadas y aplicar los aspectos requeridos [11]. Meta-AspectJ permite generar programas AspectJ sintácticamente correctos mediante plantillas de código, es una extensión de Java, por tanto es posible mezclar arbitrariamente código de Java con plantillas de código de AspectJ [12]. Javassist permite escribir meta-programas para modificar y definir clases automáticamente, simplificando la manipulación del bytecode, provee dos niveles de desarrollo: nivel de código fuente y nivel de bytecode [13].

4. Caso de estudio aplicando tecnologías Java

El caso de estudio consiste en agregar funcionalidad a un sistema con base en restricciones definidas en una especificación de requerimientos (intención). El sistema original por medio de clases y objetos representa la creación, el ordenamiento e introspección de elementos químicos con sus respectivas características. Sin embargo, en ocasiones es necesario agregar una nueva funcionalidad, dependiendo el nivel de conocimiento del alumno que utilice el sistema, dicha funcionalidad consiste en representar la creación de moléculas utilizando los elementos químicos creados por el sistema original tomando en cuenta las restricciones de composición de cada molécula. El diagrama simplificado de clases que representa la arquitectura general del sistema original se ilustra en la figura 1.

Para la implementación de la nueva funcionalidad se requieren mecanismos que permitan al usuario definir las restricciones de composición de las moléculas a generar, para esto se propone la utilización de XML (código 1). También se requiere que el sistema sea capaz de interpretar y adoptar las restricciones de composición establecidas; además es necesaria la obtención y manipulación de los elementos químicos que han sido creados por el sistema original, para esto se utiliza la exposición de contexto que ofrece AspectJ a través de su modelo de cortes. Otro mecanismo necesario es la generación e incorporación de las moléculas al sistema en ejecución, para esto se utiliza Javassist. También es necesario que las modificaciones realizadas al sistema tengan el mínimo grado de invasión al código original y que el sistema vuelva a su estado original cuando termine su ejecución, para esto se utiliza LTW. Como se muestra en la figura 2, el esquema necesario para implementar la nueva funcionalidad utiliza los tres pilares de la programación generativa (POA, definición intencional y generación de código).

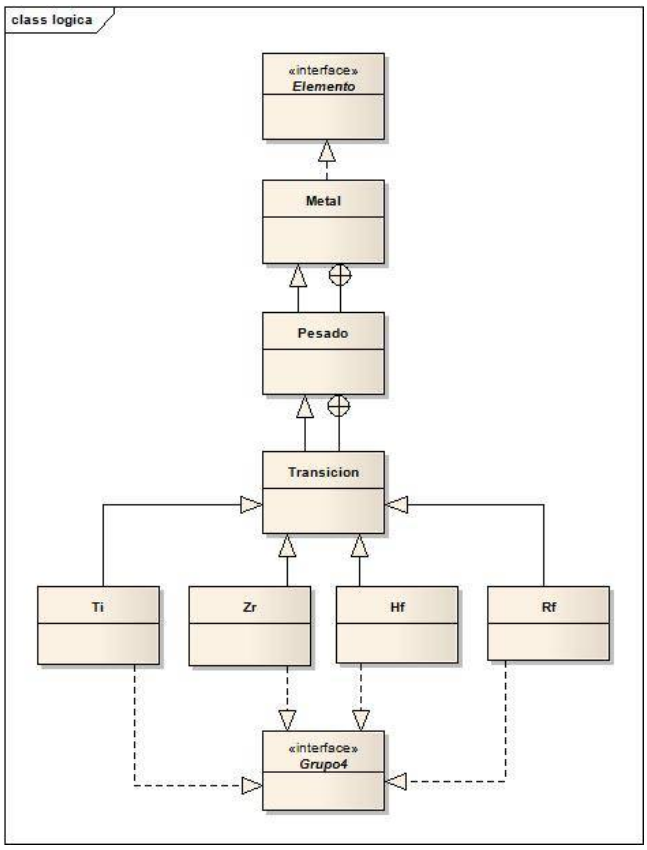

Figura 1. Fragmento del diagrama de clases del sistema original

Código 1. Restricciones de composición de la molécula de agua en el documento XML

```
<Requerimiento>
 <Componente nombre="Agua">
  <Composicion>
    <Elemento nombre="Hidrogeno" cantidad="2"
        valencia="1"></Elemento>
    <Elemento nombre="Oxigeno" cantidad="1"
        valencia="-2"></Elemento>
  </Composicion>
 </Componente>
</Requerimiento>
```

4.1. Modificación del sistema a tiempo de carga

Se opta por aplicar el entrelazado de aspectos a tiempo de carga para que la nueva funcionalidad que se va a agregar al sistema tenga la mínima intrusión en el código original y los cambios realizados no se vean reflejados cuando el sistema termine su ejecución; además de que utilizando este enfoque y mediante archivos XML es posible establecer el orden y los aspectos que serán entrelazados, permitiendo también definir aspectos concretos y configurar el weaver de AspectJ. Sin embargo es necesario tener en cuenta que al realizar los cambios

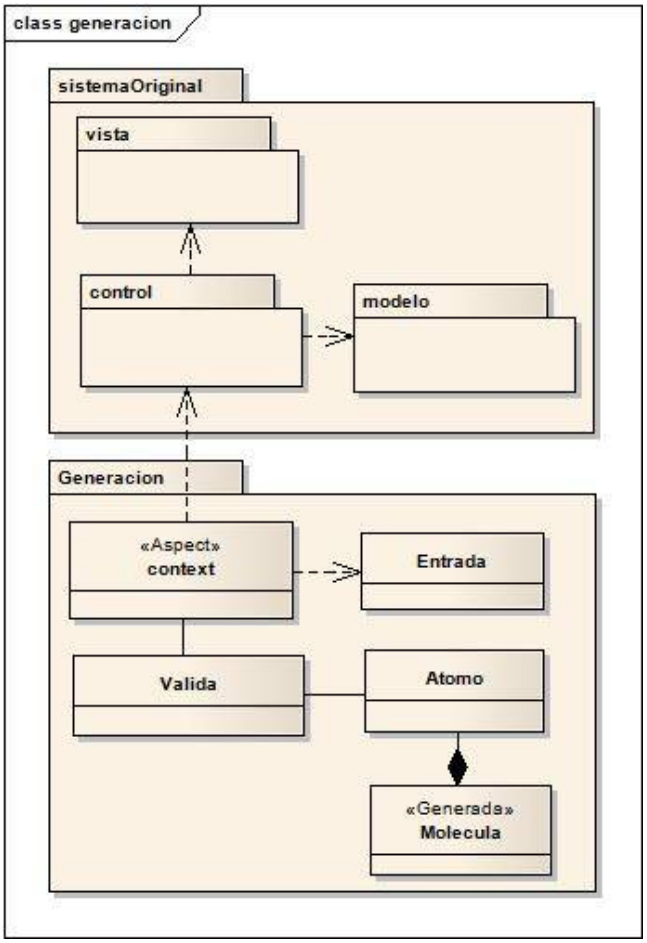

Figura 2. Diagrama de clases para el subsistema de generación de las moléculas

en tiempo de carga, no es posible agregar elementos estructurales a las clases compiladas previamente. Los pasos necesarios para realizar el entrelazado a tiempo de carga son los siguientes:

1. Compilar con ajc o javac los archivos con extensión .java que forman parte del sistema.
2. Compilar con ajc los archivos .aj que forman parte del sistema generando el XML que contiene las reglas de entrelazado y guardar la salida en un archivo .jar. (ajc -outxml -outjar “NombreDelJar” “aspecto a compilar”)
3. Ejecutar la aplicación invocando al agente, y a cada archivo .jar que contenga los aspectos a entrelazar. (java -javaagent: “ruta del agente” -classpath.;”ruta de cada archivo .jar” programa a ejecutar).

4.2. Generación mediante Javassist

La clase “Molecula” que se debe generar consta de una estructura de datos que almacena los elementos químicos recibidos por el constructor y que son mostrados por el método “getComposicion()”. El mecanismo de generación de la clase se aplica utilizando la herramienta de meta-programación Javassist, ya que el mecanismo introductions funciona únicamente para el entrelazado a tiempo de compilación y Meta-AspectJ no soporta la representación de elementos estructurales superiores a la versión 1.4 de Java. Para la generación de la clase “Molecula” se utilizan construcciones de tipo CtClass, CtField, CtConstructor y CtMethod que son representaciones que permiten definir clases, campos, constructores y métodos respectivamente. El código 2 muestra la generación de la clase molécula utilizando Javassist.

Código 2. Generación de la clase molécula mediante Javassist

```
public static void genera() throws Exception{
 ClassPool pool=ClassPool.getDefault();
 CtClass molecula=pool.makeClass("enJavassist1.
     Molecula");

 CtField campo=CtField.make("Object␣[]␣atms;",
     molecula);
 molecula.addField(campo);

 CtMethod metodo=CtNewMethod.make(""
   + "public␣void␣getComposicion()␣{"
   + "␣for(int␣i=0;␣i<atms.length;␣i++){"
   + "␣␣System.out.println(\"\"+atms[i]);"
   + "␣}"
   + "}", molecula);
 molecula.addMethod(metodo);

 CtConstructor cons=CtNewConstructor.make(""
   + "public␣Molecula(Object[]␣a){"
   + "␣atms␣=␣a;"
   + "␣getComposicion();"
   + "}", molecula);
 molecula.addConstructor(cons);

 molecula.writeFile("bin/");
}
```

5. Resultados

La trasformación del sistema se realizó en la etapa de mantenimiento, agregando la funcionalidad de una nueva clase “Molecula” a tiempo de carga por medio Javassist; del mismo modo, la definición en XML de las restricciones de composición (intención) para las moléculas a crear fue efectiva ya que se realizaron varios experimentos con distintas restricciones que resultaron exitosos. Se utilizó la biblioteca de clases JDom [14] para la interpretación del documento XML y se aplicaron mecanismos de java.util.collections para implementar

las reglas de composición definidas en el documento XML, esto por medio de las capacidades agrupamiento, parametrización e iteración de sus elementos. Javassist permitió generar y guardar en disco de manera sencilla y eficaz la nueva clase "Molecula" a la cual es posible acceder de manera natural por medio de reflection. El mecanismo LTW cumplió con el objetivo de agregar funcionalidad con la mínima intrusión al código fuente por medio de la generación a tiempo de carga de una nueva versión de la aplicación original, mostrando de esta manera el potencial para generar distintas versiones de un sistema.

6. Discusión

El entrelazado de aspectos a tiempo de carga presenta las siguientes ventajas para el desarrollador: los cambios al sistema se aplican en el bytecode que se carga a la memoria figura 3, por tanto es posible que de manera simultánea a distintos clientes se les proporcionen funcionalidades diferentes de un mismo sistema sin que éste sufra cambios; otra ventaja de este enfoque se aprecia en la facilidad de realizar cambios a un sistema estableciendo las reglas de entrelazado y definiendo nuevos aspectos mediante archivos XML. En el área de evolución de aplicaciones de software bajo el enfoque Java se alcanza a visualizar que es necesario contar con un correcto y sistematizado control de versiones de los componentes que se utilicen, modifiquen y se produzcan a lo largo del proceso de evolución de un sistema de software, esto con el objetivo de tener un control de cada cambio que se realiza a cada componente y al sistema en su totalidad permitiendo obtener una traza completa de las modificaciones realizadas. Aunado a esto se aprecia la necesidad de un lenguaje que se especialice en la generación de programas y soporte las nuevas versiones del lenguaje Java.

7. Conclusiones y trabajo a futuro

La importancia de contar con un soporte de transformación de programas radica en la necesidad de implementar de manera natural, sencilla y lo más automáticamente posible nuevos requerimientos que sean necesarios a un sistema en la fase de mantenimiento de la ingeniería de software. La generación y transformación de programas bajo el enfoque Java es posible gracias a las distintas herramientas de meta-programación que se reportan; la modificación de programas aplicando LTW es de gran utilidad para implementar diversas versiones de un mismo sistema sin alterarlo, ya que los

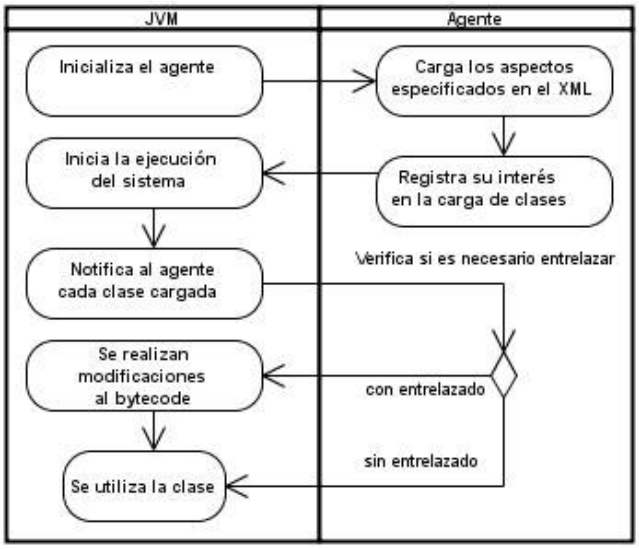

Figura 3. Proceso de entrelazado a tiempo de carga.

cambios realizados solamente se aplican cuando se ejecuta el programa, dejando al sistema original sin modificaciones. Como trabajo a futuro se considera realizar diversas versiones de sistemas distintos para apreciar otras posibles limitantes de LTW, además de buscar un mecanismo para guardar en disco las versiones generadas.

8. Agradecimientos

Este trabajo cuenta con apoyo por parte del Consejo Nacional de Ciencia y Tecnología (CONACYT).

Referencias

[1] S. D. Vermolen, G. Wachsmuth, y E. Visser, "Generating Database Migrations for Evolving Web Applications," *GPCE'11*, vol. 47, pp. 83-92, March 2012.

[2] Y. Li y G. S. Novak, "Generation of Geometric Programs Specified by Diagrams," *GPCE'11*, vol. 47, pp. 63-72, March 2012.

[3] S. Esmaeilsabzali, B. Fischer, y J. M. Atlee, "Monitoring aspects for the customization of automatically generated code for big-step models," *GPCE'11*, vol. 47, pp. 117-126, March 2012.

[4] M. Schlee y J. Vanderdonckt, "Generative Programming of graphical user interfaces," *Working conference on Advanced visual interfaces*, New York, NY. pp. 403-406, May 2004.

[5] G. Ofenbeck, T. Rompf, A. Stojanov, M. Odersky, y M. Puschel, "Spiral in scala: towards the systematic construction of generators for performance libraries," *ACM SIGPLAN Notices*, Vol. 49 pp. 125-134, March 2014.

[6] L. Passos, K. Czarnecki, S. Apel, A. Wasowski, C. Kästner, y J. Guo, "Feature-Oriented Software Evolution," *Seventh International Workshop on Variability Modelling of Software-intensive Systems*, Pisa, Italy, pp. 17-18, January 2013.

[7] T. Wurthinger, C. Wimmer, y L. Stadler, "Unrestricted and

safe dynamic code evolution for Java," *Science of Computer Programming*, Vol. 78, pp. 481-498, May 2013.

[8] V. L. Winter y A. Mametjanov, "Generative programming techniques for Java library migration," *6th International Conference on Generative Programming and Component Engineering*, Salzburg, Austria, pp. 185-196, October 2007.

[9] T. Würthinger, W. Binder, D. Ansaloni, P. Moret, y H. Mössenböck, "Applications of Enhanced Dynamic Code Evolution for Java in GUI Development and Dynamic Aspect-Oriented Programming," *ACM SIGPLAN Notices*, Vol. 46, pp. 123-126, February 2011.

[10] C. Shigeru, "Generative programming from a post object-oriented programming viewpoint," *International Conference on Unconventional Programming Paradigms*, Le Mont Saint Michel, France pp. 355-366, April 2005.

[11] R. Laddad, *AspectJ in action*, Greenwich: Manning, 2009, p. 206.

[12] S. S. Huang, D. Zook, y Y. Smaragdakis, "Domain-specific languages and program generation with meta-AspectJ", *ACM Transactions on Software Engineering and Methodology*, Vol. 18, pp. 1-32, November 2008.

[13] C. Shigeru, "Javassist a reflection-based programming wizard for Java", *OOPSLA'98 Workshop on Reflective Programming in C++ and Java*, Vancouver, Canada pp. 92-115, October 1998.

[14] JDOM project. "JDOM [ONLINE]", Noviembre 2014. http://www.jdom.org/