Programación generativa en Java y herramientas de meta-programación

José Ismael Beristain-Colorado y Ulises Juárez-Martínez

Instituto Tecnológico de Orizaba
División de Estudios de Posgrado e Investigación
Orizaba, Ver., México
isma.sp6@gmail.com, ujuarez@ito-depi.edu.mx

Resumen. La programación generativa (PG) es un paradigma de desarrollo de software el cual modela e implementa familias de sistemas de modo que es posible que un sistema se genere automáticamente con base en una especificación definida; esto con el objetivo de conseguir alta intención, reutilización y adaptación sin comprometer el desempeño en tiempo de ejecución ni los recursos del software que se produce, solventando de esta manera la necesidad de adaptación de una aplicación a nuevos requerimientos. En este artículo se describe la situación actual de la PG en general y bajo el enfoque Java; también se describe el caso de estudio de una aplicación para química, ésta aplicación representa la creación de elementos químicos por medio de la creación de objetos; a la cual, utilizando distintas herramientas de meta-programación se le agrega la funcionalidad de representar la creación de moléculas con base en los elementos creados por el sistema original.

Palabras Clave: Java, Programación Generativa, Meta-Programación.

1 Introducción

Actualmente en el proceso de desarrollo de software, la etapa de mantenimiento presenta un gran reto debido al continuo cambio en los requerimientos, si bien, la programación orientada a objetos (POO) y programación orientada a aspectos (POA) proporcionan cierto nivel de encapsulación y modificación de programas con lo cual se facilita el reemplazo y mantenimiento de piezas de software, no son suficientes para solventar la necesidad de realizar las 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, además de permitir la generación de programas con características muy similares a través de líneas de productos de software (LPS). De esta manera al tener tanta similitud, es posible mediante pequeñas modificaciones, el intercambio de las piezas

de software con que están construidos dichos programas, del mismo modo gracias al análisis de las LPS se obtiene un amplio conocimiento del comportamiento de los sistemas, permitiendo prever cuáles son las piezas que tienen más tendencia a ser modificadas y cuáles son los posibles cambios que se les aplicarán. También es posible reutilizar elementos previamente construidos para generar nuevos programas de manera automatizada. Para esto es necesario del 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 de cada una de las partes 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; a nivel de meta-programación es de gran utilidad para realizar transformaciones a las aplicaciones de software ya que permite la inserción y modificación de código usando aspectos, así como cambios a la estructura estática de los programas (introducciones).

Generadores: Un generador es un componente que obtiene como entrada una especificación (intención) de un requerimiento nuevo o cambiante y mediante distintas técnicas de meta-programación realiza de manera automática los cambios necesarios o genera nuevo código que satisfaga el cambio de requerimientos.

No obstante, a pesar de sus bondades, la programación generativa ha recibido poca atención por la industria de desarrollo de software en su aplicación práctica, ya que es poco común que en términos de desarrollo se contemple un enfoque de PG para la solución de problemas.

La contribución principal de este trabajo consiste en ilustrar mediante un caso de estudio y la utilización de diversas técnicas de meta-programación, las ventajas que presenta la implementación de distintas herramientas para la generación y transformación de programas bajo el enfoque Java; esto con el objetivo de que la industria de desarrollo de software tenga una idea más clara sobre este paradigma y lo lleve a la práctica reduciendo de esta manera el tiempo y costos invertidos a la fase de mantenimiento de software.

Este artículo está organizado de la siguiente forma: en la sección 2 se describe la situación actual de la PG desde un enfoque general. La sección 3 se refiere a la PG enfocada a las tecnologías de meta-programación que proporciona Java. La sección 4 describe el caso de estudio y muestra la implementación de cada una de las herramientas Java propuestas. La sección 5 muestra los resultados obtenidos mediante el caso de estudio. La sección 6 da pie a la discusión de ideas. Finalmente en la sección 7 se presentan las conclusiones y el trabajo a futuro.

2 Situación actual de la PG

La aplicación que se le da a la PG presenta un enfoque de autoconfiguración y generación automática de componentes, por ejemplo: en [1] se menciona que WebDSL es un lenguaje específico del dominio que provee a los desarrolladores conceptos de modelado de datos orientados a objetos con una implementación de persistencia en bases de datos relacionales; cuando el modelo de datos evoluciona, los datos necesitan migrarse, sin embargo, una migración a nivel de la base de datos rompe las abstracciones provistas por WebDSL. Por tanto se presenta un lenguaje específico del dominio para generar código que permita una evolución acoplada del modelo de datos relacional y el modelo de datos de la aplicación. Una aplicación práctica de la programación generativa se aprecia en [2] describiendo 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 basados en el razonamiento geométrico, filtra las opciones presentadas al usuario y genera el programa. En [3] se presenta un framework para monitorizar alguna propiedad del código generado en tiempo de ejecución aumentando de esta manera el nivel de abstracción con el que los desarrolladores analizan el código obtenido. En [4] se propone introducir la programación generativa en el área de generación de aplicaciones de interfaces de usuario gráficas, esto con el objetivo de demostrar que es posible generar aplicaciones personalizadas de forma automática con partes de interfaces gráficas de usuario mediante especificaciones abstractas. En [5] se mencionan los generadores de bibliotecas de alto rendimiento y las investigaciones que se han realizado 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 los 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 permitir cambios arbitrarios a definiciones de clases cargadas, permitiendo de esta manera actualizar clases en cualquier momento durante la ejecución del programa. En [8] se reporta una propuesta para la migración de bibliotecas Java mediante una herramienta que genera instancias de dichas bibliotecas, como resultado se comprueba que la generación automática de código es de gran ayuda para la migración de bibliotecas de una Máquina Virtual 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, permitiendo añadir 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*.

A continuación se describen las herramientas de meta-programación que se encuentran bajo el enfoque Java: *Meta*-AspectJ es una herramienta de meta-programación que 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* promoviendo una metodología que combina POA y PG, se utiliza para implementar generadores y de esa manera extender el poder de *AspectJ* [11]. *Jenerator* es un framework que provee mecanismos para crear clases, métodos, miembros e interfaces, permitiendo además modificar clases existentes e implementar conceptos abstractos como macros y aspectos [12]; sin embargo, *Meta-AspectJ* tiene un mayor soporte para la generación de código respecto a *Jenerator*, ya que al ser una extensión de AspectJ cuenta con todas las primitivas de corte y demás construcciones de alto nivel para la transformación de programas. *Spoonlet* es un componente de compilación distribuido como un paquete *.jar* el cual permite validar programas Java mediante análisis estático o transformar los programas mediante técnicas de PG y con base en una entrada produce código fuente adecuado para ser compilado; sin embargo, actualmente ya no se le da mantenimiento ni es compatible con versiones actuales de Eclipse [13]. *Javassist* es una herramienta de meta-programación que permite escribir meta-programas para definir clases automáticamente, simplificando la manipulación del bytecode, provee dos niveles de desarrollo: nivel de código fuente y nivel de bytecode [14].

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. La nueva funcionalidad consiste en representar la creación generativa de moléculas utilizando los elementos químicos creados por el sistema original, tomando en cuenta las restricciones de composición de cada molécula. El diagrama de clases simplificado que representa la arquitectura general del sistema original se ilustra en la Fig. 1.

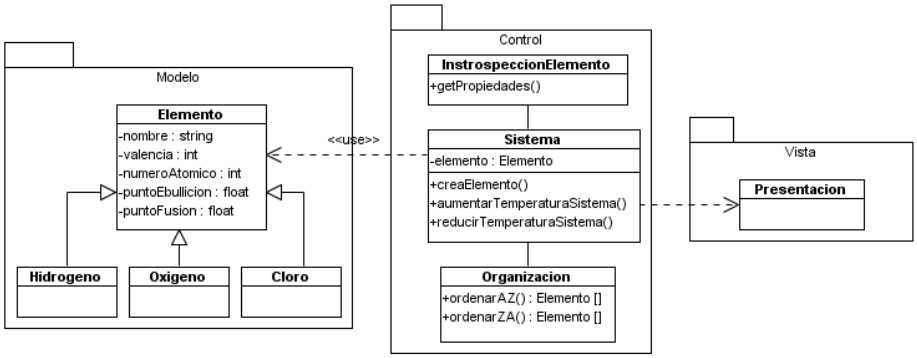

Fig. 1. Diagrama de clases del sistema original.

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 utilizan herramientas de meta-programación que permiten la generación de clases. Como se muestra en la Fig. 2, el esquema necesario para implementar la nueva funcionalidad utiliza los tres pilares de la programación generativa (POA, definición intencional y generación de código).

```
1 <Requerimiento>
2  <Componente nombre="Agua">
3   <Composicion>
4    <Elemento nombre="Hidrogeno" cantidad="2" valencia="1"></Elemento>
5    <Elemento nombre="Oxigeno" cantidad="1" valencia="-2"></Elemento>
6   </Composicion>
7  </Componente>
8 </Requerimiento>
```

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

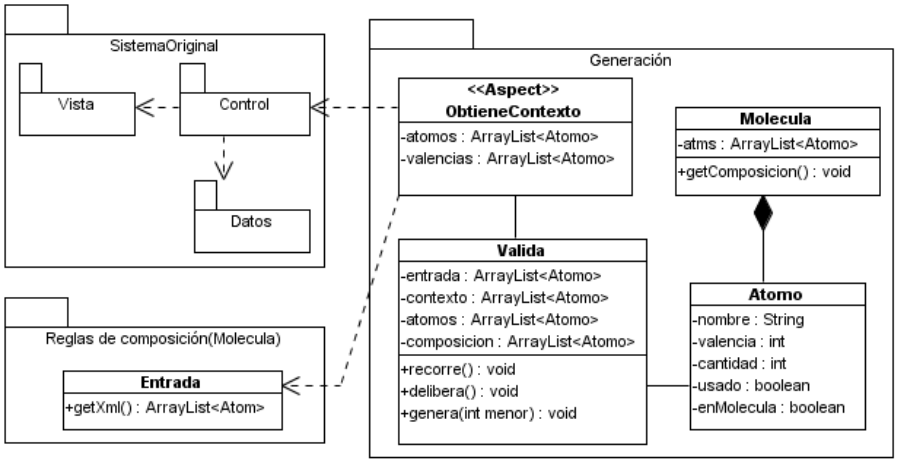

Fig. 2. Diagrama de clases para el subsistema de generación de las moléculas.

La clase "Molécula" 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 "Molecula" se aplica utilizando distintas herramientas de meta-programación: *AspectJ*, *Javassist* y *MetaAspectJ*; con el objetivo de comparar cuál es la herramienta más adecuada dependiendo el tipo de problema a abordar.

4.1 Generación mediante *AspectJ*

Utilizando únicamente los mecanismos con los que cuenta *AspectJ* no es posible generar nuevos archivos *.class* por medio de instrumentación, sin embargo, el mecanismo de *introductions* permite definir nuevos elementos estructurales a una clase existente; en este caso como se muestra en el código 2, al sistema original se le agregan el campo, el constructor, y el comportamiento necesario que debe tener la clase *Molécula*.

```
1  public aspect GeneraMolecula{
2   ArrayList<Atomo>Sistema.atms;       //introducción del campo
3   public void Sistema.getComposicion(){ //introducción del método
4       System.out.println(atms);
5   }
6   public Sistema.new(ArrayList<Atomo> a){ //introducción
7       super();                            // del constructor
8       atms = a;
9       getComposicion();
10  }
11  }
```

Código 2. Introducciones de comportamiento de una molécula al sistema original

4.2 Generación mediante *Javassist*

La biblioteca de clases *Javassist* da un amplio soporte a la generación de clases en tiempo de ejecución; para la generación de la clase "Molécula" se utilizan construcciones de tipo *CtClass, CtField, CtConstructor*y *CtMethod* que son representaciones que permiten definir clases, campos, constructores y métodos respectivamente. El código 3 muestra la generación de la clase molécula utilizando *Javassist.*

```
1  public static void genera() throws Exception{
2    ClassPool pool=ClassPool.getDefault();          //Representaciónes de:
3    CtClass molecula=pool.makeClass("enJavassist.Molecula");   // Clase
4    CtField campo=CtField.make("Object [] atms;", molecula);   // Campo
5    molecula.addField(campo);               // Añade el campo a la clase
6    CtMethod metodo=CtNewMethod.make("public void …",molecula); //Método
7    molecula.addMethod(metodo);            // Añade el método a la clase
8    CtConstructor cons=CtNewConstructor.make("public Molecul…",molecula);
9    molecula.addConstructor(cons);     // Añade el constructor a la clase
10   molecula.writeFile("bin/");        // guarda el .class en disco
11 }
```

Código 3. Generación de la clase molécula mediante *Javassist*

4.3 Generación mediante *Meta-AspectJ*

Meta-AspectJ permite crear archivos *.class* por medio de la definición de tipos de árboles sintácticos que representan elementos del lenguaje Java "*quoute* (`)"; además el elemento "*unquote* (#)" permite anexar un árbol sintáctico previamente definido a uno de mayor jerarquía, por ejemplo: añadir un método a una clase. El código 4 muestra la generación de la clase molécula utilizando *MetaAspectJ.*

```
1  public void genera() throws IOException{   // definiciones de:
2    infer pack=`[package metaAspectJ;];             // paquete
3    infer campo= `[Object [] atms;];                // campo
4    infer sentencia= `[for(int i=0; i<atms.length; i++) {…}];
5    infer metodo=`[public void getComposicion() {#sentencia}]; //método
6    infer clase=`[ #pack
7                   public class Molecula{#campo             //clase
8                     public Molecula(Object[]a){atms = a;…;}//constructor
9                     #metodo
10                  }];
11   String prog=clase.unparse();
```

```
12    FileWriter fw=new FileWriter ("C:/CasoDeEstudio/bin/metaAspectJ”);
13    fw.write(prog); fw.close();
14 }
```

Código. 4. Generación de la clase molécula mediante *MetaAspectJ*

5 Resultados

La trasformación del sistema se realizó en la etapa de mantenimiento, agregando la nueva funcionalidad, ya sea por medio de la creación de una nueva clase “Molecula” a tiempo de ejecución (*Javassist*, *MetaAspectJ*) o agregando el comportamiento necesario a una clase definida previamente (*AspectJ*); 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.

En cada uno de los experimentos con las distintas herramientas de meta-programación utilizadas se aplicó el mismo mecanismo para obtener e implementar las reglas de composición definidas en el documento *XML*.

Las tres herramientas de meta-programación con las que se realizó el caso de estudio cuentan con mecanismos de verificación de correctitud del código generado; en *AspectJ* mediante el compilador *ajc;* en *Javassist* con un compilador intermedio que verifica la correctitud de los elementos generados, y *MetaAspectJ* a través de las estructuras sintácticas que se definen para la generación de código. La Tabla 1 muestra las fortalezas que presenta cada herramienta de meta-programación en el aspecto generativo.

Tabla 1. Comparativa de herramientas de meta-programación

	AspectJ	**Javassist**	**MAJ**
Manejo de anotaciones	X	X	
Manejo de genéricos	X	X	
Representación de estructuras Java		X	X
Representaciones de estructuras de aspectos			X
Parametrización de elementos Java/AspectJ			X
Instrumentación de bytecode		X	

6 Discusión

El área de investigación y aplicación de la programación generativa es muy amplia, ya que incluye técnicas de programación orientada a aspectos, reflexión, programación intencional, generadores, componentes de software, entre otras. Sin embargo en el área de evolución de aplicaciones de software bajo el enfoque Java se alcanza a visualizar lo si-

guiente: Es necesario contar con mecanismos para la revisión y optimización del código que se modifica o genera, ya que implementar modificaciones a sistemas construidos previamente puede desencadenar en problemas a corto o largo plazo en términos de rendimiento o funcionalidad. También 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.

Como se aprecia en la Fig. 3, la transformación de programas aplicando técnicas de PG en el proceso de desarrollo de software se realiza en la etapa de mantenimiento. Al presentarse requerimientos nuevos o cambiantes, se realizan las modificaciones necesarias o se construyen nuevos elementos en tiempo de ejecución para agregar la nueva funcionalidad esperada realizando la menor cantidad de cambios posibles al sistema original.

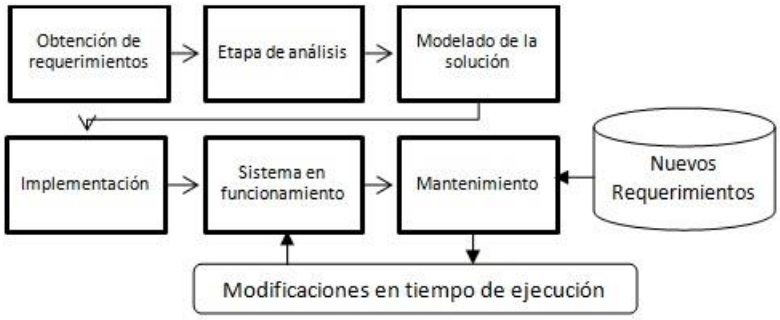

Fig. 3. Transformación de programas en el ciclo de vida de software.

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, sin embargo, es necesario contar con un mayor soporte que permita al programador una implementación transparente y lo más automatizada posible.

Los resultados obtenidos permiten detectar las capacidades de cada herramienta de meta-programación presentada, con esto es posible detectar, dependiendo la situación, la

herramienta o conjunto de herramientas necesarias para la transformación y generación de programas.

Como trabajo a futuro se considera realizar otros casos de estudio controlados y puntuales en la generación de programas para determinar los alcances de cada herramienta de meta-programación en el desarrollo de software.

Agradecimientos

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

Referencias

1. Vermolen Sander D, Wachsmuth Guido and VisserEelco "Generating Database Migrations for Evolving Web Applications" in GPCE'11, Oregon, USA, pp. 83-92. (2011)
2. Li Yulin and Novak Jr. Gordon S."Generation of Geometric Programs Specified by Diagrams" in GPCE'11, Oregon, USA, pp. 63-72.(2011)
3. Esmaeilsabzali Shahram, Fischer Bernd and Atlee Joanne M. "Monitoring aspects for the customization of automatically generated code for big-step models" in 10th ACM international conference on Generative programming and component engineering, New York, USA, pp. 117-126 (2011)
4. Max Schlee and Jean Vanderdonckt, "Generative Programming of graphical user interfaces," in proceedings of the Working conference on Advanced visual interfaces, New York, pp. 403-406, ISBN:1-58113-867-9 (2004)
5. Ofenbeck Georg [et al.]"Spiral in Scala: Towards the Systematic Construction of Generators for Performance Libraries" in GPCE '13, Indianapolis USA, pp. 125-134 (2013)
6. Passos Leonardo, Krzysztof Czarnecki, Apel Sven, Wasowski Andrezj et al, "Feature-Oriented Software Evolution" in proceedings of The Seventh International Workshop on Variability Modelling of Software-intensive Systems, Italy (2013)
7. Thomas Wurthinger, Christian Wimmer, and Lukas Stadler, "Dynamic code evolution for Java," in proceedings of the 8th International Conference on the Principles and Practice of Programming in Java, pp. 10-19, New York (2010)
8. Victor L. Winter and Azamat Mametjanov, "Generative programming techniques for Java library migration," in proceedings of the6th international conference on Generative programming and component , 185 – 196, New York (2007)
9. Wurthinger Thomas [et al.] Applications of Enhanced Dynamic Code Evolution for Java in GUI Development and Dynamic Aspect-Oriented Programming in GPCE'10, pp. 123-126, Eindhoven, Holanda, (2010)
10. Chiba Shigeru, "Generative programming from a post object-oriented programming viewpoint," in proceedings of the international conference on Unconventional Programming Paradigms, pp. 355-366, Berlin (2004)
11. Huang Shan Shan, Zook David and Smaragdakis Yannis "Domain-specific languages and program generation with meta-AspectJ" in ACM Trans. Softw. Eng. and Methodology, Vol. 18-2, pp. 32, New York, USA, November (2008)

12. Volter M. and Gartner A. "Jenerator - Generative Programming for Java" in OOPSLA Workshop on Generative Programming, Tampa Bay, Florida (2001)
13. INRIA. Tutorial: Enhancing Java with Spoon. [Online]. http://spoon.gforge.inria.fr/TutorialJDT/TutorialJDT, (2013)
14. Chiba Shigeru, "Javassist—a reflection-based programming wizard for Java" in OOPSLA'98 Workshop on Reflective Programming in C++ and Java, pp. 92-115, Vancouver, BC, Canada (1998)