CIENCIA *ergo-sum*
Universidad Autónoma del Estado de México
ciencia.ergosum@yahoo.com.mx
E-ISSN: 2395-8782

Panorama general de la Programación Generativa

Hernández Martínez, Juan Alberto

Panorama general de la Programación Generativa
CIENCIA *ergo-sum*, vol. 26, núm. 1, marzo-junio 2019 | **e46**
Universidad Autónoma del Estado de México, México

Hernández Martínez, J. A. (2019). Panorama general de la Programación Generativa. *CIENCIA ergo-sum, 26*(1). https://doi.org/10.30878/ces.v26n1a11

Panorama general de la Programación Generativa

An Overview of Generative Programming

Juan Alberto Hernández Martínez
Instituto Tecnológico de Orizaba, México
*jhernandezm@ito-depi.edu.*mx

Recepción: 8 de mayo de 2018
Aprobación: 11 de julio de 2018

Resumen

Automatizar el desarrollo de *software* es un reto difícil de lograr debido a la creciente diversidad tecnológica ocasionada principalmente por la inclusión de diferentes dominios y el cumplimiento de nuevas necesidades por parte de los usuarios. Por tanto, se brinda un panorama general de la Programación Generativa destacando sus objetivos, principios básicos, los requerimientos para su adopción en la industria del *software* y la tecnología disponible en la actualidad para abordarla. La literatura muestra que el paradigma generativo es extenso y poco utilizado en la industria. Sin embargo, representa una alternativa atractiva para fabricar de forma automatizada productos personalizables desde especificaciones de alto nivel que permitan pasar de la producción de sistemas simples a la producción de familias de sistemas.

Palabras clave: Programación Generativa, ingeniería de *software*, desarrollo de *software* automatizado, familias de sistemas.

Abstract

Software development automation is a hard challenge due to the increasing technological sophistication and diversity, caused mainly by the inclusion of new domains, as well as new users' need fulfillment. In this paper, an overview of Generative Programming is provided, highlighting its main goals, basic principles, requirements for its adoption in the software development industry, and the available technology to address it. The literature shows that nowadays the generative paradigm is very wide and little addressed in the software industry. However, it represents an attractive alternative to implement highly customizable and optimized intermediary or final products from high-level specifications in an automated manner, changing the focus from single systems software development to families of systems.

Keywords: Generative Programming, Software Engineering, Automated Software Development, Family of Systems.

Introducción

La computación se ha posicionado como un área de apoyo muy importante para la mayoría de los sectores industriales existentes, ya que permite agilizar y realizar tareas complejas en menor tiempo y con menos esfuerzo (Shoemaker *et al.*, 2016). Sin embargo, ser piedra angular de muchos sectores y organizaciones representa un gran reto, pues es necesario contar con los mecanismos adecuados para hacer frente tanto al ritmo vertiginoso de desarrollo de productos como al constante cambio tecnológico que día a día incrementa de forma exponencial.

Ante esta situación, la ingeniería de *software* se ha encargado de establecer diferentes procesos para implementar productos de calidad tratando de reducir tiempo, esfuerzo y costos en la elaboración de cada uno de ellos. No obstante, aun con todo el potencial que posee actualmente, se logra detectar que es necesario contar con elementos que permitan alcanzar una producción automatizada de *software*, de tal manera que toda la diversidad tecnológica sea solventada de forma rápida y con menos esfuerzo tal y como algunas otras ingenierías llevan a cabo sus procesos de producción (Vogel-Heuser *et al.*, 2014).

En este sentido, la Programación Generativa es una área de la computación que permite modelar familias de sistemas de *software* y ayuda a la fabricación de productos altamente personalizables u optimizados a partir

de la especificación de un requerimiento de alto nivel mediante el uso de generadores, con lo cual se logra un beneficio en el desarrollo de *software* con mejoras en sus capacidades de adaptación, reutilización y evolución (Batory, 2004; Radošević *et al.*, 2011). Aunado a ello, es importante remarcar que la implementación y utilización de generadores trae consigo beneficios que impactan no sólo en la calidad y cantidad de productos elaborados sino también en la disminución de esfuerzo, tiempo, costo, adaptación y evolución de componentes de *software*.

En este artículo se brinda un panorama general del paradigma de la Programación Generativa destacando los objetivos que persigue y que la convierten en una alternativa atractiva para el desarrollo de *software* hacia las masas. Además, se describen los puntos necesarios para su adopción y es también un primer acercamiento de su extensión, de la que se resaltan las áreas más relevantes que la conforman y las herramientas más sobresalientes para abordarla.

1. Fundamentos del paradigma generativo

La Programación Generativa es un paradigma computacional cuya filosofía se basa en el modelado de familias de sistemas de *software*. De manera general, su intención es diseñar e implementar módulos de *software* que puedan ser combinados para generar sistemas especializados y altamente optimizados a través de generadores que satisfagan un requerimiento en particular (Czarnecki, 2005; Beristain-Colorado y Juárez-Martínez, 2014).

2. Objetivos y principios de la programación generativa

De manera específica, el paradigma generativo tiene cuatro objetivos concretos: *a*) lograr una alta intencionalidad, *b*) una alta reutilización y adaptación, *c*) simplificar la administración de muchas variantes de un componente y *d*) aumentar la eficiencia tanto en espacio como en tiempo de ejecución.

Para abordar de manera adecuada cada uno de los objetivos anteriores, la Programación Generativa se apoya de diferentes principios tales como la separación de asuntos, parametrización de diferencias, análisis y modelado de dependencias e interacciones, separación del espacio del problema del espacio de la solución, así como eliminar la sobrecarga y realizar optimizaciones específicas de un dominio.

Con el fin de entender el propósito, la interrelación e importancia de cada punto anterior en el contexto generativo, a continuación se describen los elementos más relevantes que los conforman.

2. 1. Lograr una alta intencionalidad

La Programación Generativa brinda un medio flexible para reducir la brecha conceptual entre el código de un programa y los conceptos de dominio, es decir, lograr que el código fuente de un programa sea lo suficientemente expresivo como para capturar la intención exacta de lo que el desarrollador tiene en mente cuando concibe su trabajo. Para conseguir esta funcionalidad, la Programación Generativa se apoya de tres enfoques principales: *a*) programación intencional, *b*) técnicas de lenguajes de dominio específico y *c*) metaprogramación.

Por un lado, dentro del paradigma de la Programación Intencional desarrollado por Charles Simonyi, las *intenciones* representan un elemento clave. Estas abstracciones permiten reflejar el comportamiento específico del *software* tal y como el desarrollador lo tiene en mente: basándose en el entendimiento del lenguaje natural. Además, permiten que los desarrolladores rompan la barrera del "cómo" para centrarse en el "qué" (Simonyi *et al.*, 2006; Beristain-Colorado y Juárez-Martínez, 2014), lo cual minimiza de manera considerable el tiempo y esfuerzo empleado por los desarrolladores en actividades de implementación de código.

Por otro lado, para mejorar la funcionalidad de dichas *intenciones*, se emplean técnicas de lenguajes de dominio específico que básicamente corresponden a tareas de optimización y verificación de errores y también metaprogramación para implementar extensiones de lenguajes de programación en caso de ser necesario. El resultado final de combinar estos enfoques es una serie de vistas que permiten editar el código fuente utilizando representaciones más cercanas al significado lógico que a la gramática del lenguaje de programación.

Con el fin de entender claramente la información anterior, la tabla 1 presenta la sumatoria de todos los números comprendidos entre 1 y 10.

Tomando en cuenta la información de la tabla 1, se aprecia que de forma tradicional el programador se centra en cómo solucionar el problema pensándolo en términos de un lenguaje de programación, es decir, en el código que satisfaga cada una de las condiciones de dicha problemática, mientras que de manera intencional sólo se limita a expresar de manera exacta qué desea conseguir y lo especifica en términos de un lenguaje de programación cuya expresividad es más cercana al lenguaje natural.

TABLA 1
Ejemplo de código intencional

Código tradicional	Código intencional

`int s = 0;`
 	
`for(int i=1; i<=10; i++){`
`s += i;`
`}`
	“add the numbers from 1 to 10”

Fuente: elaboración propia.

2. 2. Alta reutilización y adaptación a través de la separación de asuntos

La Programación Generativa adopta el principio de *separación de asuntos* como una alternativa atractiva para beneficiar la reutilización y adaptación de componentes. En términos simples, la separación de asuntos permite descomponer un programa en diferentes partes de tal manera que cada una trate con un asunto importante a la vez, es decir, con un requisito de interés para todos los involucrados en un proyecto (Tsang *et al.,* 2004; Santos *et al*., 2016).

Para llevar esta filosofía a la implementación de código, la Programación Generativa se apoya en el paradigma de la programación orientada a aspectos, la cual esta encargada de proveer de manera natural los mecanismos adecuados al programador para lograr que cada decisión se tome en un lugar concreto y evitar que el código de un programa tenga que tratar con diferentes asuntos de manera simultánea.

Básicamente, el mecanismo utilizado por la programación orientada a aspectos consta de tres elementos: *a*) un lenguaje base tal como Java, C++, C#, etcétera., con el cual se implementan los componentes, *b*) un lenguaje de aspectos tal como AspectJ, el cual posee el código que se encuentra diseminado por la estructura de otras unidades funcionales y *c*) un entrelazador que se encarga de producir el sistema final combinando el código base con el código de aspectos (Hernández-Martínez y Juárez-Martínez, 2012).

Con el fin de entender de mejor manera los conceptos anteriores, la tabla 2 presenta la sumatoria de todos los números comprendidos entre 1 y 10 resaltando las diferencias entre el desarrollo tradicional y con orientación a aspectos.

TABLA 2
Implementación de programas con el paradigma de aspectos

Forma tradicional	Programación orientada a aspectos
Código en lenguaje de programación Java	Código base utilizando el lenguaje de programación Java

```
public class Sumatoria {

 //Código para sumatoria de números
 public int sumatoria() {

 //Línea de código para desplegar mensaje inicial
 System.out.println("Sumatoria de números del 1 al 10");

 int s = 0;

 for(int i=1; i<=10; i++){
 s += i;
 }

return s;
 }

 public static void main(String... args) {
 Sumatoria obj = new Sumatoria();

 //Línea de código para imprimir el resultado
 System.out.println("Resultado: " + obj.sumatoria());
}
}
```

```
//Código únicamente para sumatoria de números
public class Sumatoria {

public int sumatoria() {
int s = 0;

for(int i=1; i<=10; i++){
s += i;
}

return s;
}

public static void main(String... args) {
new Sumatoria().sumatoria();
}
}
```

Código de Aspectos utilizando el lenguaje AspectJ

```
//Mensaje inicial e impresión del resultado
public aspect Aspecto {

pointcut imprimir():execution(* Sumatoria.sumatoria());

before(): imprimir() {
 System.out.println("Sumatoria de números del 1 al 10");
}

after() returning (int s):imprimir() {
System.out.println("Resultado: " + s);
}
}
```

Fuente: elaboración propia.

Con base en la tabla 2, se aprecia que de forma tradicional la solución al problema está dada por un sólo código que incluye la impresión de un mensaje inicial, la sumatoria de números y la impresión del resultado de dicha operación. Aunque en términos de implementación es totalmente válido, este tipo de prácticas resulta contraproducente debido a que ocasiona problemas típicos de código enmarañado, código disperso o código repetido.

Por otro lado, se aprecia que desde la perspectiva de la separación de asuntos, el desarrollador descompone el programa en secciones: una con el fragmento de código que soluciona la sumatoria de números y otra que contiene el comportamiento que no corresponde a la lógica de negocio principal y que suele diseminarse a lo largo del programa. En el contexto del ejemplo planteado corresponden a las líneas de código que dan vida al mensaje inicial de la sumatoria y a la impresión del resultado.

Es necesario mencionar que el uso de la programación orientada a aspectos no sólo mejora la legibilidad y limpieza del código sino que también brinda a los desarrolladores una mayor facilidad para razonar sobre los conceptos, ya que se encuentran separados y tienen una dependencia mínima, lo que a su vez mejora el nivel de reutilización, adaptación y mantenimiento de los programas.

2. 3. Parametrización de diferencias y análisis y modelado de dependencias e interacciones

Un aspecto relevante en el desarrollo de *software* basado en componentes, además de lograr una correcta modularidad en las aplicaciones, es cómo lograr que los componentes existentes puedan ser aprovechados de mejor manera para desarrollar aplicaciones de forma más flexible.

Para solventar esta situación, la Programación Generativa emplea dos enfoques. El primero de ellos es la parametrización de diferencias, la cual permite representar lo que se denominan *familias de componentes*, es decir, componentes que poseen muchas características en común. La relevancia de este enfoque se debe a que provee un medio adecuado para que los desarrolladores tengan a su disposición toda una gama de componentes base de tal manera que puedan crear diferentes aplicaciones; además, resulta una alternativa atractiva para reducir el tiempo y esfuerzo en su desarrollo.

El segundo enfoque se trata de un análisis y modelado de dependencias e interacciones, el cual hace referencia a todas aquellas combinaciones válidas o no válidas entre los valores de parámetros que cada componente requiere para su funcionamiento, así como también las dependencias que existirían entre dichos valores, ya que en ocasiones los valores de algunos parámetros pueden implicar los de algunos otros.

Vale la pena resaltar que ambos enfoques son parte fundamental para el direccionamiento del desarrollo de *software* hacia la producción de familias de sistemas con el fin de satisfacer las necesidades de sectores de mercado completos y no sólo para clientes únicos.

2. 4. Separación del espacio del problema del espacio de la solución

Si recordamos un poco, en la Programación Generativa es importante mantener una correcta separación de asuntos. Esta filosofía se debe llevar no sólo a niveles de código sino también a un nivel más abstracto. Para ello, en el contexto generativo es necesario separar el espacio del problema del espacio de la solución.

En términos simples, el espacio del problema consiste en todas aquellas abstracciones de un dominio específico con las cuales interactúa un desarrollador de *software*, es decir, requerimientos o características de una aplicación específica.

Por otro lado, el espacio de la solución contiene componentes de implementación o decisiones de diseño que pueden ser aplicadas para crear instancias de un *software* resultante. Un ejemplo de esto son los componentes genéricos, los cuales son alimentados con valores específicos para obtener un componente final con una funcionalidad bien definida.

Aunque el espacio de la solución y el espacio del problema tienen una estructura diferente debido al tipo de información que contienen, ambos mantienen una comunicación mediante el uso de configuraciones automáticas que a menudo requieren de técnicas de metaprogramación. De esta manera, a partir de un requerimiento se puede dar paso a la implementación de componentes o sistemas concretos.

2. 5. Eliminación de sobrecarga y realización de optimizaciones específicas de un dominio

La existencia de problemas tales como código sin utilizar o sentencias que realizan verificaciones en tiempo de ejecución, por mencionar algunos, es algo común y suele ocasionar sobrecarga en el funcionamiento del *software*. Para abordar esta situación, la Programación Generativa opta por generar componentes en tiempo de compilación, así como optimizaciones específicas.

3. Adopción del paradigma generativo

Para adoptar el paradigma generativo se requiere principalmente realizar una transición hacia la fabricación automatizada de *software* y para ello es necesario tomar en cuenta dos aspectos: *a*) cambiar el enfoque de ingeniería de sistemas simples a ingeniería de familias de sistemas y *b*) automatizar el ensamblado de componentes de *software* utilizando generadores.

3. 1. De la ingeniería de sistemas simples a la ingeniería de familias de sistemas

Durante mucho tiempo la ingeniería de *software* se ha orientado hacia el desarrollo de *software* a la medida en donde cada sistema es fabricado desde cero con base en las especificaciones, requerimientos y necesidades de un usuario particular. Aunque este tipo de desarrollo resulta adecuado para muchas organizaciones, es poco flexible al momento de cubrir las necesidades de nuevos usuarios, ya que todas las funcionalidades de los sistemas fabricados están centradas en satisfacer los requerimientos de usuarios individuales, lo que conlleva a reducir las posibilidades de lograr una diversificación de productos.

Dada esta situación, la Programación Generativa propone dirigir el enfoque hacia las familias de sistemas, es decir, hacia la fabricación de productos a gran escala teniendo como piedra angular a las líneas de productos de *software*.

De manera formal, una línea de productos de *software* se define como un conjunto de sistemas de *software* intensivo que comparten un conjunto de características comunes y administradas, las cuales satisfacen necesidades específicas de un segmento de mercado particular y que son desarrolladas de forma prescrita a partir de un conjunto común de activos base (*Software Product Lines*, 2017).

Una forma tangible de entender este concepto es mediante el esquema de una línea de producción de automóviles, la cual posee una plataforma base con todos los componentes que resultan comunes en su fabricación, es decir, todos aquellos que se encuentran sin importar su modelo, tamaño y demás propiedades y, por otro lado, una gama de componentes que definen las características particulares para cada uno, de tal manera que la combinación de estos últimos con los componentes comunes den como resultado un automóvil específico.

3. 2. Automatizando el ensamblado de componentes mediante generadores

En el contexto de la Programación Generativa los generadores son componentes responsables de producir sistemas de *software* desde especificaciones de alto nivel, mejor conocidas como *intenciones*.

Para lograr su objetivo, los generadores emplean una estrategia que consta de cuatro actividades: *a*) recibir una especificación de requerimiento o intención, *b*) verificar si el sistema o componente solicitado puede ser construido, *c*) completar la especificación mediante el cálculo de valores predefinidos y *d*) ensamblar los componentes necesarios para obtener un sistema o componente concreto (Hernández-Martínez *et al*., 2018).

La importancia de los generadores radica principalmente en que representan el medio para reducir la brecha entre el alto nivel de abstracción, las descripciones de sistemas intencionales y el sistema ejecutable deseado.

4. Funcionamiento general

De acuerdo con la información reportada en secciones anteriores, podemos decir de manera simple y concreta que el paradigma generativo permite a los programadores indicar lo que quieren mediante una intención; de este modo, un componente generador se encarga de producir el sistema o componente deseado de forma automatizada, siempre con miras hacia la producción de familias de sistemas (figura 1).

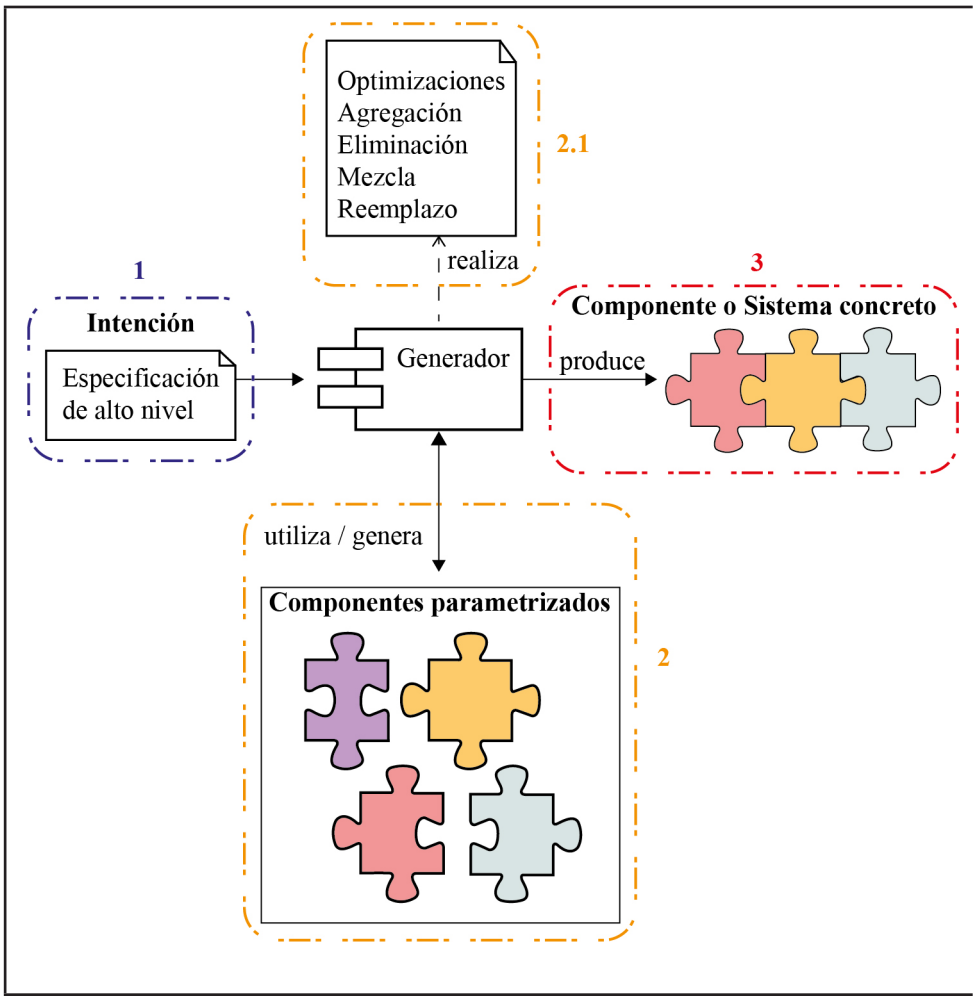

FIGURA 1
Funcionamiento de un componente generador
Fuente: elaboración propia.

5. Extensión del paradigma generativo

Con base en la información descrita en los apartados anteriores, se alcanza a apreciar que el paradigma generativo es extenso de manera relativa. Abordarlo significa adentrarse en un mundo lleno de áreas, técnicas, principios y diversas tecnologías que poseen un nivel de complejidad significativamente alto.

Para tener un primer acercamiento a su dimensión, la figura 2 presenta los enfoques más relevantes del paradigma generativo.

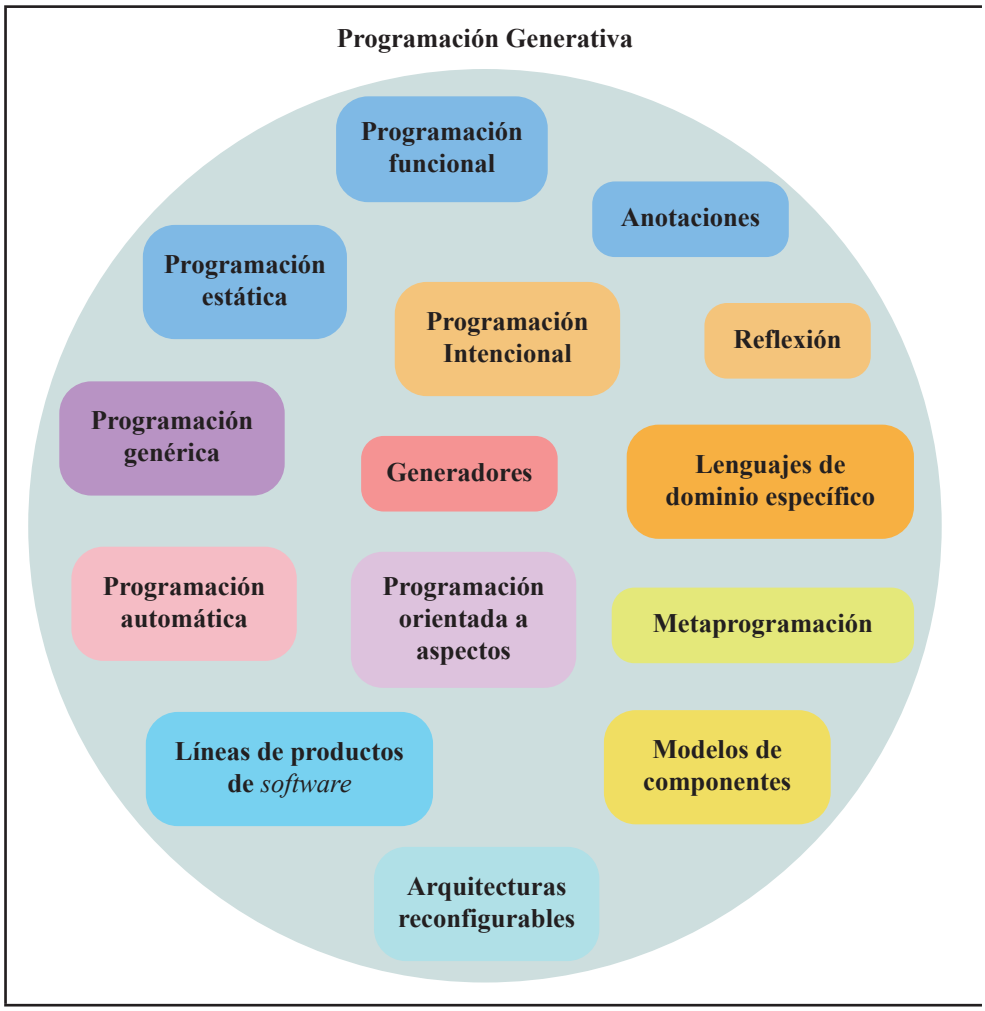

FIGURA 2
Algunas áreas involucradas en la Programación Generativa
Fuente: elaboración propia.

6. Herramientas, lenguajes y tecnología del paradigma generativo

Un punto importante a tener en cuenta respecto al paradigma generativo es que, a pesar de la gran diversidad de enfoques tecnológicos que involucra, la Programación Generativa tiene como pilares a tres áreas: *a*) programación orientada a aspectos, *b*) programación intencional y *c*) generadores; dichas áreas concentran el fundamento de la generatividad y fungen como punto de partida para abordar algunas otras más puntuales de manera directa o indirecta.

En este sentido, la tabla 3 presenta algunas de las tecnologías más sobresalientes y utilizadas para abordar la generatividad con base en cada uno de los pilares mencionados.

Cabe mencionar que hasta el momento no se reporta una herramienta que aborde el paradigma generativo de manera completa. Sin embargo, la comunidad científica computacional sigue realizando esfuerzos de manera constante para madurar cada uno de los enfoques existentes, en especial en cuestiones de generación de comportamiento, modularidad y manejo de variabilidad de componentes de forma dinámica (Vogel-Heuser *et al.,* 2015).

TABLA 3
Tecnologías más utilizadas en el paradigma generativo

Área	Tecnología	Propósito	Funcionamiento general
Generadores	DRACO (Tripathy y Naik, 2014)	Enfoque para la ingeniería de dominio basada en lenguajes de dominio específico y tecnología de transformación.	Permite organizar el conocimiento de construcción de *software* en una serie de dominios relacionados, en donde básicamente cada dominio encapsula el conocimiento para resolver cierta clase de problemas.
	GenVoca (Javed *et al.,* 2016)	Enfoque para la construcción de generadores de *software* basados en la composición de capas de abstracción orientadas a objetos.	De manera concreta, GenVoca permite apilar capas una sobre otra, las cuales contienen un número de clases que refinan las capas inferiores mediante la agregación de nuevas clases, métodos, etcétera.
Programación orientada a aspectos	AspectJ (Valente *et al.*, 2010; Eclipse, 2018)	Extensión del lenguaje Java para encapsular asuntos de corte junto con las reglas de entrelazado necesarias para coordinarse con los objetos de un sistema.	De manera general, AspectJ permite definir aspectos, los cuales constan de cortes (colecciones de eventos en la ejecución de un programa) y avisos que especifican las acciones que se van a desarrollar cuando se identifiquen ciertos puntos de unión, es decir, los lugares donde los cortes se llevan a cabo.
Programación intencional	Intentional Platform (Intentional, 2017)	Plataforma para desarrollar y ejecutar aplicaciones intencionales.	Básicamente, la plataforma trabaja con aplicaciones intencionales que encapsulan conocimiento como un activo tangible que las computadoras pueden procesar.
	DSLs (Intentional, 2017)	Lenguajes de dominio específico para guardar o registrar el conocimiento.	El conocimiento se registra a través de un editor de proyección mediante lenguajes de dominio específico integrados. Una vez registrado, queda disponible para su procesamiento hasta transformarlo en nuevas aplicaciones de *software*.
	Meta (Intentional, 2017)	Tecnología intencional utilizada de tres formas: metaprogramación, metalenguaje y metaaplicación.	De manera general, Meta provee un lenguaje de alto nivel de abstracción que permite crear intenciones para un dominio o contexto particular y a partir de ellas generar programas concretos.

Fuente: elaboración propia.

Conclusiones

La filosofía del paradigma generativo requiere que la implementación de componentes se ajuste a una arquitectura de líneas de productos común a través de generadores, y sobre todo contar con un mecanismo que permita indicar la forma de traducir requerimientos abstractos dentro de una gama de componentes específicos.

Es necesario puntualizar que la Programación Generativa implica dos ciclos de desarrollo: *a*) para reutilizar, en donde los componentes son planificados, diseñados y sistematizados hacia la obtención de familias de sistemas y *b*) con reutilización, en donde los insumos disponibles son utilizados para producir sistemas concretos.

Finalmente, el enfoque de desarrollo generativo evita que las aplicaciones evolucionen de forma oportunista; además, produce un cambio natural cuando se trabaja con componentes de *software*, ya que, en lugar de buscar los componentes necesarios para producir un sistema concreto, éstos son generados.

Análisis prospectivo

Más allá de la complejidad que pueda representar la Programación Generativa, resulta un paradigma muy interesante y con muchas bondades para aquellas empresas que logran incorporarlo.

En primer lugar obtienen un acercamiento hacia la forma en cómo otras industrias fabrican sus productos como empresas de manufactura, líneas de producción de automóviles, alimentos, componentes electrónicos, por mencionar algunas. De igual manera, al emplear técnicas generativas logran reducir la intervención humana y por consecuencia el esfuerzo empleado al fabricar sistemas, lo cual resulta atractivo para hacer frente al ritmo acelerado que el desarrollo de aplicaciones actual demanda.

Si bien por extensión y complejidad la Programación Generativa podría estar destinada hacia empresas de desarrollo grandes que poseen una infraestructura robusta, las pequeñas y medianas empresas también pueden incursionar e incorporar su filosofía. Esta situación les permitiría posicionarse de manera estratégica dentro del sector computacional como empresas con niveles de producción en masa, capaces de cubrir un sector de mercado completo.

Es importante considerar que la información reportada en este artículo está dirigida a todos aquellos interesados e involucrados en el área computacional, desde estudiantes hasta desarrolladores y arquitectos de *software*. Sin embargo, aquellas personas que son dueñas de empresas desarrolladoras pueden encontrar en esta información una base para iniciar el direccionamiento de sus organizaciones hacia nuevos horizontes, así como también un escalamiento de tal manera que les permita adquirir nuevos niveles de madurez y alcances.

Agradecimientos

Se agradecen los comentarios de los árbitros de la revista.

Referencias

Batory, D. (2004). The road to utopia: A future for generative programming. *Lecture notes in computer science*, 1-18.

Beristain-Colorado, J. I. y Juárez-Martínez, U. (2014). Programación generativa en Java y herramientas de meta-programación. *Advances in Intelligent Information Technologies*, 37-47.

Czarnecki, K. (2005). Overview of Generative Software Development, en J.-P. Banâtre, P. Fradet, J.-L. Giavitto y O. Michel (eds.), *Unconventional Programming Paradigms* (pp. 326-341).Springer.

Eclipse, F. (2018). *The AspectJ Project*. Disponible en https://www.eclipse.org/aspectj/

Hernández-Martínez, J. A., Juárez-Martínez, U. y Cardozo, N. (2018). Sýntixi–A generative approach to dynamic fusion of software components, en M. Genero y M. Kalinowski (eds.), *Proceeding of the XXI Ibero-American Conference on Software Engineering* (*CIbSE 2018*). Bogotá: Universidad de Los Andes.

Hernández-Martínez, J. A. y Juárez-Martínez, U. (2012). Anfibio-Herramienta para visualización de estructuras de datos utilizando aspectos de grano fino. *Segundo Congreso Internacional de Computación México-Colombia* (pp. 210-217). Chilpancingo.

Intentional. (2017). The Intentional Platform. Disponible en http://www.intentsoft.com/intentional-technology/

Javed, M., Naeem, M., Umar, A. I. y Bahadur, F. (2016). Automated inconsistency detection in feature models: A generative programming based approach. *Selforganizology*, *3*(2), 59-74.

Radošević, D., Magdalenić, I., y Orehovački, T. (2011). Error messaging in generative programming, en T. Hunjak, S. Lovrenčić y I. Tomičić (Eds.), *Proceedings of the 22nd Central European Conference on Information and Intelligent Systems (CECIIS 2011)*. Varaždin: University of Zagreb.

Santos, A., Alves, P., Figueiredo, E., y Ferrari, F. (2016). Avoiding code pitfalls in aspect-oriented programming. *Science of Computer Programming*, *119*, 31-50.

Shoemaker, D., Woody, C. y Mead, N. R. (2016). Advances in software engineering and software assurance. *Advances in Computers*, *102*, 1-46. Elsevier.

Simonyi, C., Christerson, M. y Clifford, S. (2006). Intentional Software. *Proceedings of the 21st annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications*, *41*(10), 451-464.

Software Product Lines. (2017). Disponible en https://www.sei.cmu.edu/productlines

Tripathy, P. y Naik, K. (2014). *Software evolution and maintenance: A practitioner's approach*. John Wiley & Sons.

Tsang, S. L., Clarke, S. y Baniassad, E. (2004). An evaluation of aspect-oriented programming for java-based real-time systems development. *Proceedings of the Seventh IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, 291-300.

Valente, M. T., Couto, C., Faria, J. y Soares, S. (2010). On the benefits of quantification in AspectJ systems. *Journal of the Brazilian Computer Society*, *16*(2), 133-146.

Vogel-Heuser, B., Diedrich, C., Fay, A., Jeschke, S., Kowalewski, S., Wollschlaeger, M. y Göhner, P. (2014) Challenges for software engineering in automation. *Journal of Software Engineering and Applications*, *7*, 440-451. 10.4236/jsea.2014.75041.

Vogel-Heuser, B., Feldmann, S., Folmer, J., Ladiges, J., Fay, A., Lity, S.,...y Lamersdorf, W. (2015). Selected challenges of software evolution for automated production systems. *Industrial Informatics* (*INDIN*), *2015 IEEE 13th International Conference on Industrial Informatics*, 314-321.