# Un acercamiento práctico a la programación políglota

Kedwin Johan Pérez Zea y Raquel Anaya - Corporación Universitaria Adventista

## I. Introducción

Actualmente, el mundo del desarrollo de software está centrado en el uso de un lenguaje estándar para la implementación de los proyectos; con lenguajes de programación como Java, PHP o C#, son escritas la mayoría de las aplicaciones que se producen en la industria [1]. La principal razón por la cual predominan las monoculturas es que un lenguaje estándar de programación es fácil de usar, tanto para los desarrolladores como para los administradores cuando de conseguir personal para el proyecto se trata.

Consecuentemente, se ha visto la necesidad de introducir frameworks que facilitan el trabajo a los desarrolladores, en funcionalidades que cada vez requieren mayor especialización, como el acceso a base de datos, el manejo eficiente de XML, la construcción rápida de interfaces de usuarios o la creación de servicios web, ya que el lenguaje por sí solo lo hace un poco tedioso. Estos nuevos frameworks introducen abstracciones y normalmente requieren un extenso conocimiento para su configuración y uso, elevando la curva de aprendizaje y la complejidad de la solución [4].

Usar un solo lenguaje de programación pasa por alto el hecho de que los seres humanos tienden a usar diferentes lenguajes para hacer que las expresiones sean más efectivas. Por lo tanto, si la implementación se realiza de forma más natural en otro lenguaje de programación, en lugar de usar un nuevo framework, podría resultar una solución más viable, ya que aquel es mucho más cercano al problema que se está intentando resolver.

A este enfoque se le denomina programación políglota, y su objetivo es ofrecer soluciones más sencillas, más expresivas y fluidas, combinando las mejores soluciones de diferentes lenguajes y paradigmas de programación.

Este artículo se basa en las investigaciones realizadas por algunos académicos y expertos que han sembrado las bases para este

relativamente nuevo tema de estudio [2] [3] [4].

En este artículo se pretende exponer los conceptos en los cuales se fundamenta esta nueva tendencia del campo del desarrollo de software. Se pretende además, mostrar algunos ejemplos de implementaciones usando este acercamiento en algunas fases del desarrollo, evidenciando las ventajas que trae el uso de esta práctica para atender las diferentes demandas de una solución software. El interés en este tema, por parte del autor principal, surge de manera natural al enfrentarse por primera vez a un entorno industrial de producción de software en su experiencia de pasantía empresarial.

## II. Aspectos fundamentales de la programación políglota

Según Watts, la programación políglota es la actividad de usar varios lenguajes de programación en un solo sistema informático de software, de igual modo que los humanos usan diferentes lenguajes para hacer más efectivas las expresiones con las que se comunican. El objetivo de la programación políglota es ofrecer soluciones más sencillas, combinando las mejores soluciones de diferentes lenguajes de programación [10].

Hasta ahora, algunas aplicaciones de la programación políglota han sido, por ejemplo, en el campo de los compiladores, donde se facilita la creación de estos para lenguajes similares a Java, a la vez que se ataca el problema del código duplicado [5]; también ha sido utilizada en el campo de la genética, donde se han desarrollado aplicaciones para el análisis de datos genéticos, procesando grandes volúmenes de información con algoritmos complejos [6]. El alto rendimiento y la flexibilidad que este tipo de soluciones demanda, se alcanzan usando múltiples lenguajes de programación. Otra aplicación muy común y natural de la programación políglota es el desarrollo de aplicaciones web, la cual se analiza con mayor detalle en la siguiente sección.

Aunque el término es relativamente nuevo, la idea de programación políglota no lo es; dos principales razonamientos han llevado a plantearse este tema [2].

El primer razonamiento es que no existe la bala de plata, es decir, no existe una herramienta que sea la mejor para resolver todos los problemas [7]; se debe encontrar la herramienta cercana al problema que se quiera resolver; en el desarrollo de software implica examinar y seleccionar los lenguajes, frameworks y herramientas que se usarán para la implementación de la solución.

Fowler afirmó que sería más conveniente buscar otro lenguaje en el cual el problema sea resuelto de forma más natural que introduciendo un nuevo framework construido en el mismo lenguaje, ya que el costo del aprendizaje del nuevo lenguaje sería equivalente al del aprendizaje del nuevo framework, pero este haría mejor trabajo, expresando la solución de forma más efectiva [9].

El segundo razonamiento es que los desarrolladores son más costosos que el hardware, y por ello la productividad de los desarrolladores se ha convertido en una de las principales consideraciones dentro del proceso de desarrollo de software. Una de las principales premisas de la programación políglota es lograr el incremento de la productividad de los desarrolladores, combinando e integrando las mejores soluciones de diferentes lenguajes de programación, logrando de esta manera una solución más simple, fácil de construir y de mantener en el tiempo [4].

III. Programación políglota en el desarrollo web ajustar espacio amplio

El desarrollo web busca dividir la estructura de las aplicaciones en capas, separando la vista, el control, la lógica de negocios y el acceso a datos, intentando respetar principios arquitectónicos como separación de intereses y bajo acoplamiento y alta cohesión [16]. El acelerado surgimiento de nuevos lenguajes, como respuesta a la creciente demanda de productividad en el desarrollo, ha derivado en que las aplicaciones web modernas tienden a ser sistemas políglotas, que usan diferentes lenguajes de programación y especificaciones [11].

Un enfoque monocultural sería utilizar un framework que satisface todas las necesidades de cada capa, ofreciendo un conjunto de APIs que convierten el código del lenguaje estándar, en el lenguaje que los navegadores son capaces de interpretar, en el caso de la vista. Para el caso de la lógica de negocios y las operaciones transversales a la aplicación, el framework presta ciertas facilidades para el desarrollo, necesitando de antemano cierto conocimiento sobre este para su correcta configuración e implementación. Igualmente, cuando de la persistencia y de acceso a base de datos se trata, el framework especializa y extiende las funcionalidades del lenguaje, reduciendo la verbosidad de las operaciones, pero finalmente agregando más abstracciones al código.

En un enfoque políglota, la capa de presentación es implementada directamente en uno de estos frameworks javascript que actualmente están teniendo tanta acogida en el desarrollo web [18], los cuales generalmente implementan el patrón Modelo Vista Controlador (MVC), combinándolo con el concepto de Data Binding, el cual permite la sincronización del modelo con la vista sin tener que manipular el DOM manualmente [19].

La capa central de la aplicación es implementada en un lenguaje estable, de alto rendimiento, multiparadigma, donde se pueda expresar de forma elegante y concisa la lógica de negocio, gracias a que generalmente estos lenguajes combinan características de los lenguajes funcionales y de los lenguajes orientados a objetos [20]. En esta capa

también se proponen introducir los DSL , los cuales son lenguajes que, como su nombre lo indica, se enfocan en representar una parte específica del dominio. Estos logran que el código sea más fácil de comprender por su gran expresividad, lo cual mejora la mantenibilidad de la solución; también, pueden llegar a ser entendidos por las personas del negocio, permitiéndoles comprender directamente el código que implementa sus reglas de negocio [21]. Para el acceso a datos comúnmente se introducen los ORM . Estos permiten consultar y manipular la base de datos encapsulando el código necesario para hacer esto con un enfoque orientado a objetos, y permitiendo interactuar directamente con un objeto o tupla del lenguaje que se esté utilizando [22].

En el siguiente cuadro comparativo (Tabla I) se ejemplifica este planteamiento, mostrando una solución comúnmente propuesta en la industria desde un enfoque monocultural utilizando tecnologías Java, y otro desde un enfoque políglota.

Tabla I
Arquitectura por capas.
Enfoque monocultural y políglota

| **Capa** | **Propuesta Monocultural** | **Propuesta Políglota** |
|---|---|---|
| **Vista** | JSF, Spring web flow | AngularJs, React |
| **Lógica de negocios** | Spring beans, Annotations, Transactions | Scala, Ruby on Rails, DSL's |
| **Acceso a datos** | SQL String Query | Squeryl, ActiveData |

IV. Aplicaciones políglotas en el ciclo del desarrollo de software

A continuación se describen algunas de las fases del ciclo de desarrollo en donde la programación políglota hace un aporte efectivo.

A. Desarrollo

Como se justificó anteriormente, la productividad de los desarrolladores es un aspecto bastante importante en el ciclo del desarrollo del software, y es precisamente la principal razón por la cual ha aumentado la popularidad de la programación políglota en el desarrollo web, introduciendo frameworks creados usando lenguajes de programación dinámicos, los cuales prometen aumentar la productividad

proporcionando valor de negocio más rápido [12].

Usualmente estos frameworks son basados en la filosofía de Convención sobre Configuración , la cual pretende reducir el número de decisiones que los desarrolladores deben tomar al momento de usar estos framework [17]. Los más importantes de estos son: Grails basado en Groovy, Ruby on Rails basado en Ruby y Django basado en Python.

La programación políglota es útil si esta trae alguna ventaja al momento del desarrollo. Por ejemplo, una sobrecarga causada por el análisis y la conversión de archivos XML puede ser removida si un lenguaje de programación con soporte literal para XML como Groovy o Scala es usado en vez de un lenguaje de propósito general con una librería de manejo XML. Este mismo principio también se aplica para el manejo de JavaScript Object Notation (JSON) si el lenguaje soporta esta notación nativamente [4].

En las siguientes figuras se compara código utilizado para cargar y procesar un archivo XML tanto en Java como en Scala, respectivamente. Se puede observar cómo el uso de librerías es Java para procesamiento de archivos XML, es mucho más complejo mientras que el procesamiento en Scala se hace de manera más natural.

Figura 2. Procesamiento XML en Java. Tomado ajustar de https://www.tutorialspoint.com/java_xml/java_dom_parse_document.htm

```
File inputFile = new File("input.txt");
DocumentBuilderFactory dbFactory
   = DocumentBuilderFactory.newInstance();
DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
Document doc = dBuilder.parse(inputFile);
doc.getDocumentElement().normalize();
System.out.println("Root element :"
   + doc.getDocumentElement().getNodeName());
NodeList nList = doc.getElementsByTagName("student");
System.out.println("----------------------------");
for (int temp = 0; temp < nList.getLength(); temp++) {
   Node nNode = nList.item(temp);
   System.out.println("\nCurrent Element :"
      + nNode.getNodeName());
   if (nNode.getNodeType() == Node.ELEMENT_NODE) {
      Element eElement = (Element) nNode;
      System.out.println("Student roll no : "
         + eElement.getAttribute("rollno"));
      System.out.println("First Name : "
         + eElement
         .getElementsByTagName("firstname")
         .item(0)
         .getTextContent());
      System.out.println("Last Name : "
      + eElement
         .getElementsByTagName("lastname")
         .item(0)
         .getTextContent());
      System.out.println("Nick Name : "
      + eElement
         .getElementsByTagName("nickname")
         .item(0)
         .getTextContent());
      System.out.println("Marks : "
      + eElement
         .getElementsByTagName("marks")
         .item(0)
         .getTextContent());
   }
}
```

Figura 1. Procesamiento XML en Scala. Tomado ajustar de http://bcomposes.com/2012/05/04/basic-xml-processing-with-scala/

```
val musicElem = scala.xml.XML.loadFile("/tmp/music.xml")

(musicElem  "artist").foreach { artist =>
  println((artist  "@name").text + "n")
  val albums = (artist  "album").foreach { album =>
    println("  " + (album  "@title").text + "n")
    val songs = (album  "song").foreach { song =>
      println("     " + (song  "@title").text)
    }
  println
  }
}
```

B. Testing

El testing o pruebas es un concepto fundamental en las metodologías de desarrollo ágil, incluyendo el desarrollo guiado por pruebas (TDD)

y extreme programming (XP), las cuales han hecho más popular la automatización de las pruebas en los proyectos [13] [14].

Las pruebas son una forma perfecta de introducir la programación políglota en el ciclo de desarrollo del software, ya que aquellas no son una parte integral de la aplicación, pero deben cubrir cada aspecto de ella. Automatizar las pruebas les brinda la confianza a los desarrolladores de que los nuevos cambios que sean introducidos al sistema no afectarán los desarrollos anteriores. Esto es especialmente crítico en los lenguajes de programación dinámicos, donde debido a la falta de tipos estáticos, el código no se descompila si se introduce un cambio que rompa el sistema [15].

Adicionalmente, hacer programación políglota no implica descartar lo que ya se ha implementado. Los beneficios de lenguajes de programación mejor equipados para el tema de las pruebas pueden aprovecharse incluso en arquitecturas ya existentes.

En las siguientes figuras se comparan dos pruebas unitarias de un mismo método Java, una escrita en el mismo lenguaje del framework para mocking JMock, y la otra escrita en JRuby. Puede verse, igualmente, que la solución en Java es mucho más compleja mientras que en JRuby resulta más natural.

Figura 3. Prueba unitaria escrita en Java utilizando JMock, Framework para mocking de objetos.

```
1  public class OrderInteractionTester extends MockObjectTestCase {
2      private static String TALISKER = "Talisker";
3
4      public void testFillingRemovesInventoryIfInStock() {
5
6          Order order = new OrderImpl(TALISKER, 50);
7          Mock warehouseMock = new Mock(Warehouse.class);
8
9          warehouseMock.expects(once()).method("hasInventory")
10             .with(eq(TALISKER),eq(50))
11             .will(returnValue(true));
12         warehouseMock.expects(once()).method("remove")
13             .with(eq(TALISKER), eq(50))
14             .after("hasInventory");
15
16         order.fill((Warehouse) warehouseMock.proxy());
17
18         warehouseMock.verify();
19         assertTrue(order.isFilled());
20     }
21
22 }
```

Figura 4. Prueba unitaria a código Java escrita en JRuby.

```
1  class OrderInteractionTest < Test::Unit::TestCase
2      TALISKER = "Talisker"
3
4      def test_filling_removes_inventory_if_in_stock
5          order = OrderImpl.new(TALISKER, 50)
6          warehouse = Warehouse.new
7          warehouse.stubs(:hasInventory).with(TALISKER, 50).returns(true)
8          warehouse.stubs(:remove).with(TALISKER, 50)
9
10         order.fill(warehouse)
11         assert order.is_filled
12     end
13
14 end
```

## V. Conclusiones

Benjamin Whorf, un lingüista estadounidense, afirma que el lenguaje influye directamente en la forma de pensar del individuo. Según Whorf, el lenguaje tiene una incidencia directa con respecto a lo que la persona puede pensar y lo que puede hacer.

Este razonamiento es también aplicable al contexto del desarrollo de software y los lenguajes de programación: si un desarrollador conoce un solo lenguaje de programación, estará limitado a pensar y resolver problemas solo basado en este lenguaje. La forma como un desarrollador Java piensa, es totalmente diferente de cómo uno de Ruby piensa, y ambos solucionarían el mismo problema de manera diferente

gracias a la influencia recibida por el lenguaje que cada uno conoce [23].

Se trata entonces de un cambio de cultura de la nueva generación de desarrolladores: Un desarrollador con cultura políglota es aquel que conoce y usa de forma efectiva varios lenguajes de programación, lo que le permite expresar sus pensamientos de distintas formas, ver los problemas desde diferentes puntos de vista y dar soluciones con diversas abstracciones.

La programación políglota pretende brindar soluciones basadas en lo mejor de cada lenguaje y cada plataforma, buscando desarrollar sistemas cada vez más robustos, más flexibles y mejor posicionados para cumplir las demandas modernas [24].

Es muy importante el papel que la academia debe cumplir en la formación de los desarrolladores de software; se deben crear espacios de aprendizaje donde se promueva el uso de diversos lenguajes, desarrollando en el estudiante la competencia de seleccionar cuál es el lenguaje más apropiado para cada necesidad particular de una solución software.

Referencias números sin negrillas


1. Cass, S. (2015). The 2015 top ten programming languages. IEEE Spectrum, July, 20.
2. Fjeldberg, H. C. (2008). Polyglot programming (Doctoral dissertation, Master thesis, Norwegian University of Science and Technology, Trondheim/Norway).
3. Keznikl, J., Malohlava, M., Bures, T. & Hnetynka, P. (2011, August). Extensible Polyglot Programming Support in Existing Component Frameworks. In Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on (pp. 107-115). IEEE.
4. Harmanen, J. (2013). Polyglot Programming in Web Development.
5. Nystrom, N., Clarkson, M. R. & Myers, A. C. (2003, April). Polyglot: An extensible compiler framework for Java. In International Conference on Compiler Construction (pp. 138-152). Springer Berlin Heidelberg.
6. Nowak, R. M. (2014). Polyglot programming in applications used for genetic data analysis. BioMed research international, 2014.
7. Hans-Christian Fjeldberg (2008). Polyglot Programming, A business perspective. Norwegian University of Science and Technology. Department of Computer and Information Science.
8. Brooks, F. P. (1987, April). No silver bullet: Essence and accidents of software engineering. Computer, 20(4), 10–19.

9. Fowler, M. (2007a). One language. Retrieved 2017-03-20, from http://martinfowler.com/bliki/OneLanguage.html
10. Watts, N. (2008). Even more than polyglot programming. Retrieved 2008-05-11, from http://thewonggei.wordpress.com/2008/01/22/even-more-than-polyglot-programming/
11. Ford, N. 2006. Polyglot Programming. [WWW]. [Accessed on 02.12.2012]. Available at: http://memeagora.blogspot.com/2006/12/polyglot-programming.html
12. Bachle, M. & Kirchberg, P. (2007). "Ruby on rails". Software, IEEE, 24(6), 105-108. DOI 10.1109/BCI.2009.31.
13. Janzen, D. and Saiedian, H. 2005. Test-driven development concepts, taxonomy, and future direction. IEEE Computer, 38(9), 8 p.
14. Beck, K. 1999. Embracing Change with Extreme Programming. IEEE Computer, 32(10), 8 p.
15. Abadi, M., Cardelli, L., Pierce, B. & Plotkin, G. (1991). Dynamic typing in a statically typed language. ACM transactions on programming languages and systems (TOPLAS), 13(2), 237-268.
16. Taylor, R. N., Medvidovic, N. & Dashofy, E. M. (2009). Software architecture: Foundations, theory, and practice. Wiley Publishing.
17. Bächle, M. & Kirchberg, P. (2007). Ruby on rails. IEEE Software, 24(6).
18. Gizas, A., Christodoulou, S. & Papatheodorou, T. (2012, April). Comparative evaluation of javascript frameworks. In Proceedings of the 21st International Conference on World Wide Web (pp. 513-514). ACM.
19. Darwin, P. B. & Kozlowski, P. (2013). AngularJS web application development. Packt Publ.
20. Odersky, M., Altherr, P., Cremet, V., Emir, B., Maneth, S., Micheloud, S. & Zenger, M. (2004). An overview of the Scala programming language (No. LAMP-REPORT-2004-006).
21. Fowler, M. (2008). Domain Specific Language. Retrieved 2017-04-17, from https://martinfowler.com/bliki/DslQandA.html
22. Stoltmann, T. Object-relational Mapping.
23. Whorf, B. L. & Chase, S. (1956). Language, Thought and Reality, Selected Writings of Benjamin Lee Whorf. Edited by John B. Carroll. Foreword by Stuart Chase. J. B. Carroll (Ed.). Mass.
Bonér, J., Farley, D., Kuhn, R. & Thompson, M. (2014). The reactive manifesto.

Fecha de recepción: 23 de julio de 2018.

Fecha de aprobación: 25 de julio de 2018.

Kedwin Johan Pérez Zea - Corporación Universitaria Adventista
Correo electrónico:

johanperez@unac.edu.co

Raquel Anaya - Corporación Universitaria Adventista
Correo electrónico: raquel.anaya.hdez@gmail.com