

El Desarrollo del Framework Orientado al Objeto

por [Marcus Eduardo Markiewicz](#) y [Carlos J.P. de Lucena](#)
Traducido por [PauloN. Lama](#)

Introducción

Los frameworks orientados al objeto (llámense simplemente frameworks) son la piedra angular de la moderna ingeniería del software. El desarrollo del framework está ganando rápidamente la aceptación debido a su capacidad para promover la reutilización del código del diseño y el código fuente (source code). **Los frameworks** son los Generadores de Aplicación que se relacionan directamente con un dominio específico, es decir, con una familia de problemas relacionados.

Como ejemplo, considere la construcción de un kit de herramientas de interface gráfica del usuario (GUI Tool Kit). Puede ser que elijamos diseñar y poner un solo kit de herramientas en ejecución. Por otra parte, si diseñamos el kit de herramientas como framework, este puro diseño nos permitirá generar una colección de los kits de herramientas para una variedad de aplicaciones del tipo GUI (Graphic User Interface). Los frameworks deben generar las aplicaciones para un dominio entero. Por lo tanto, debe haber puntos de flexibilidad que se puedan modificar los requisitos particulares para ajustarse a la aplicación. Por ejemplo, un punto de extensión puede ser el algoritmo usado para trazar elementos gráficos.

Los puntos flexibles de un framework se llaman **los puntos calientes (hot-spots)**. Los puntos calientes o Hot-spots son las clases o los métodos abstractos que deben ser implementados o puestos en ejecución. Los frameworks no son ejecutables. Generar un ejecutable, uno debe "instanciar" el framework (llámese Instanciar, al hecho de producir y completar un objeto llenando con valores en lugar de variables en un class template) poniendo el código específico de la aplicación en ejecución para cada punto caliente. Una vez que los puntos calientes sean "instanciados", el framework utilizará aquellas classes usando el callback o repetición de la llamada (acto de repetir la autenticación del número de usuario en caso de reconexión). En esta repetición de la llamada o callback, el código del usuario del servicio declara que desea ser llamado en la ocurrencia un determinado evento. Entonces, el código del proveedor del servicio realiza la repetición de la llamada o callback con el código del usuario del servicio al momento de ocurrir ese determinado evento. Por esta razón, en primera instancia, el framework se caracteriza a veces como "el viejo código que llama al nuevo código."

Algunas de las características del framework no son mutables ni tampoco pueden ser alteradas fácilmente. Estos puntos inmutables constituyen el núcleo o kernel de un framework, también llamados como **los puntos congelados o frozen-spots** del framework. A diferencia de los puntos calientes o hot-spots, los puntos congelados o inmutables son los pedacitos del código puestos en ejecución ya dentro del framework que llaman a uno o más puntos calientes proporcionados por el ejecutor. El núcleo o **Kernel** será la constante y presentará siempre la parte de cada instancia del framework

Piense en un framework como si fuese un motor. Un motor requiere potencia. A diferencia de un motor tradicional, un motor del framework tiene muchas entradas de potencia. Cada uno de estas entradas de potencia es un punto caliente del framework. Cada punto caliente debe ser accionado para que el motor funcione. Los generadores de potencia son el código específico de la aplicación que se debe enchufar a los puntos calientes. El código agregado de la aplicación será utilizado por el código kernel del framework. El motor no correrá hasta que todos los enchufes estén conectados. Esta metáfora se ilustra en el cuadro 1.

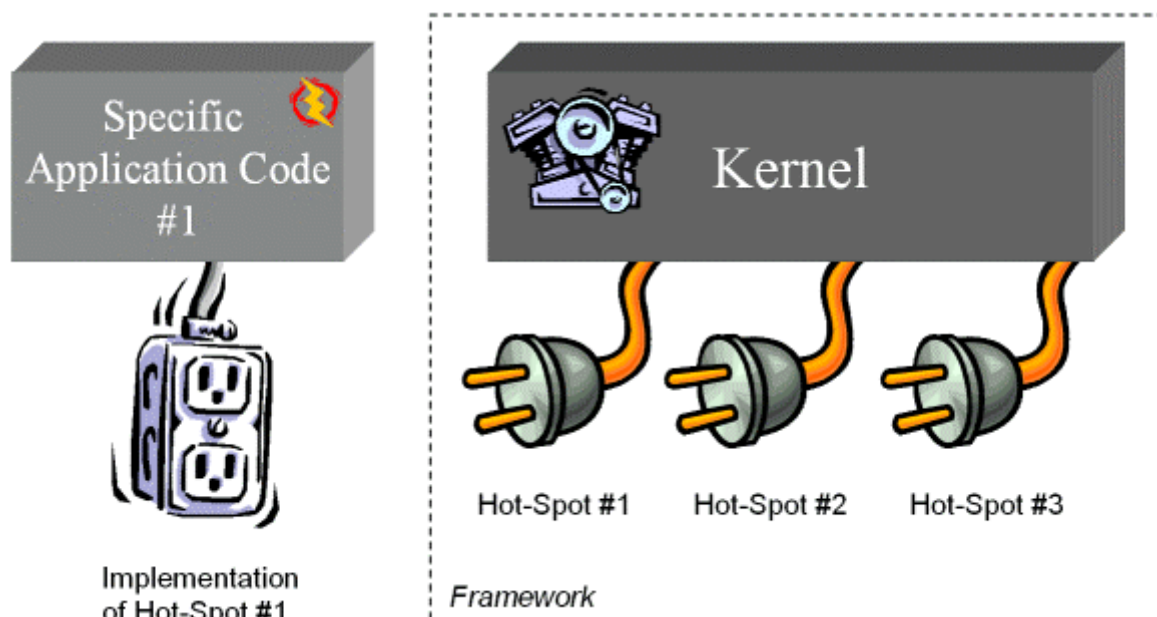


Figura 1. Frameworks

La capacidad de reutilización del código y del diseño de frameworks orientados al objeto permite una productividad mayor y un tiempo de Mercado breve en el desarrollo de aplicaciones, en comparación con el desarrollo tradicional de los sistemas de software [4]. La configuración flexible de frameworks, permite la reutilización del núcleo kernel. El desarrollo del framework ha sido exitoso en muchos dominios. Algunos ejemplos incluyen el Microsoft Foundation Classes (**MFC**) framework, el Object Management Group's (**OMG**) [3] y el COM+ y DCOM de Microsoft [11]. Para clarificar frameworks, repasaremos algo de los framework de prueba, el JUnit.

Entendiendo los Frameworks: JUnit, el framework de prueba.

JUnit [7] es un framework simple, bien diseñado, de prueba para JAVA. La configuración de JUnit se muestra como un diagrama de la clase de UML class en el Cuadro 2 [1]. Cada rectángulo representa una clase (class). La sección superior lleva su nombre y en la parte inferior sus métodos. Cada relación entre estas clases es representado por las barras que las conectan. La barra de flechas que conecta las clases de TestCase y Test Classes indica que "es una" relación, siguiendo la dirección de la flecha; así, la clase de TestCase hereda de la Test Class. La barra que conecta entre las clases de TestResult y de TestCase indica una " asociación, " es decir las llamadas de una clase los otros métodos. La línea que señala una punta de diamante indica que " tiene una " relación, donde la clase más cerca a la punta de diamante es la única que tiene la otra; en este ejemplo, la clase de TestSuite tiene un ejemplo o caso de una de las clases derivadas de Test Class (las clases TestSuite y TestCase).

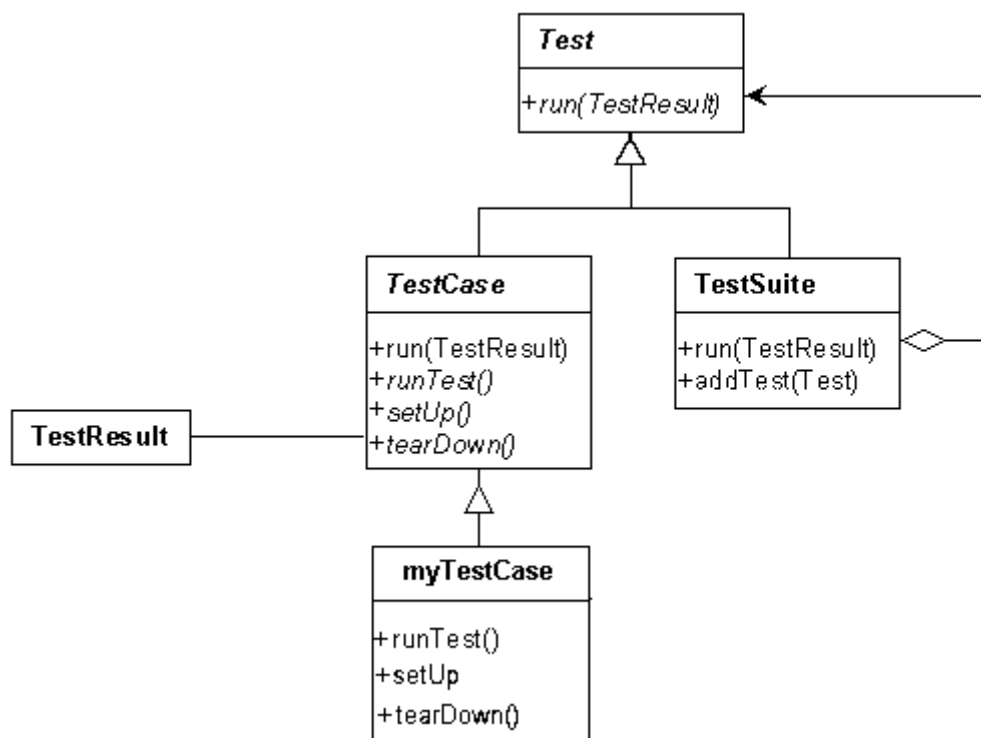


Figura 2. Las clases de framework de JUnit

Se proporciona el framework con un "cookbook" o libro de recetas, que describe gradualmente, el proceso de "instanciación". Por ejemplo, este framework puede ser instanciado poniendo la clase de TestCase en ejecución. El método del run(), heredado de la clase Test ejecutará una prueba de un caso ejecutando el setUp(), el runTest(), y el tearDown() en secuencia. Explicaremos el papel de estos métodos cuidadosamente.

El método del runTest() lleva a cabo llamadas de la función API (Application Program Interface), el método del llamado, y otro código a probar. El framework de JUnit también proporciona la infraestructura para ejecutar dos o más pruebas en conjuntos similares o idénticos de objetos. Esto se llama base de la prueba, y es posible con los métodos del setUp() y del tearDown(). El método del setUp() inicializa las variables del objeto. Correspondientemente, el método del tearDown() borra los recursos asignados por el setUp(). Cada vez que se llama al método del run(), se ejecuta el método del setUp() que configura el escenario de prueba, la prueba es ejecutada, entonces tearDown() destruye el escenario creado. El generador de la aplicación "instancializa" el framework a través de la clase myTestCase, el cual subsecuentemente, "instancializa" los métodos discutidos anteriormente más arriba.

Las cajas de la prueba se pueden también agrupar en compartimientos de prueba (TestSuite). Un compartimiento de prueba determinado puede contener varios casos de prueba o varios compartimientos de prueba. Cuando el método del run() sea invocado para un compartimiento de prueba, todos sus pruebas son ejecutados recurrentemente. Los resultados de las pruebas se recogen en las clases de TestResult, que contienen los detalles de las pruebas falladas. Se agrupan usando patrones de diseño compuesto [5].

Los puntos calientes de este framework son el código de prueba, la construcción del escenario de la prueba, y el código de destrucción. Todos estos son implementados por los métodos descritos arriba. Incluso más, este framework tiene una interfaz muy clara y definida, donde la prueba de clase abstracta, es extendida por el TestCase y el TestSuite.

Desarrollo del Framework

Las tres etapas principales del desarrollo del framework son análisis del dominio, diseño del framework, y la "instanciación" del framework.

El análisis del dominio procura descubrir los requisitos del dominio y los posibles requerimientos futuros. Para completar los requerimientos sirven las experiencias previamente publicadas, los sistemas de software similares existentes, las experiencias personales, y los estándares considerados. Durante el análisis del dominio, los puntos calientes y los puntos congelados se destapan parcialmente.

La fase del diseño del framework define las abstracciones de éste. Se modelan los puntos calientes y los puntos congelados (quizás con diagrama de UML, Modelo Unificado del Lenguaje, *Unified Modeling Language* [1]), y la extensión y la flexibilidad propuesta en el análisis del dominio se esboza en líneas generales. Según lo mencionado arriba, los modelos del diseño se utilizan en esta fase.

Finalmente, en la fase de "instanciación", los puntos calientes del framework son implementados, generando un software del sistema. Es importante observar que cada uno de estas aplicaciones tendrá los puntos congelados del framework en común. Las fases del proceso del desarrollo del framework son comparados con las tradicionales fases del diseño orientados al objeto, según se puede apreciar en el cuadro 3. En esta figura, nombramos las fases del desarrollo según lo descrito. [6].

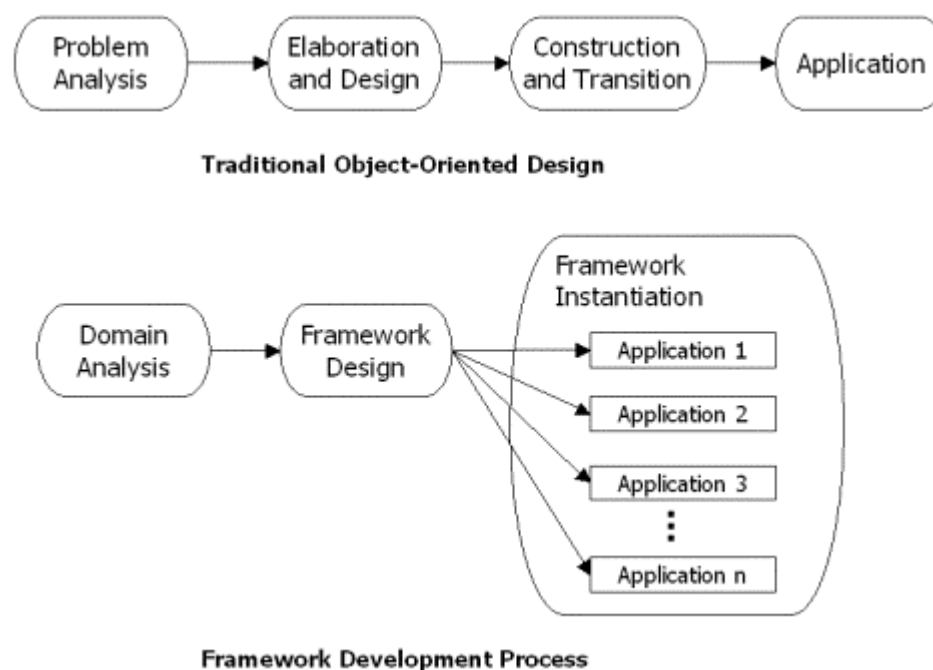


Figura 3.Proceso de desarrollo del framework

Según lo mostrado en el cuadro 3, el desarrollo tradicional Orientado al Objeto se diferencia del desarrollo del framework. En el desarrollo tradicional Orientado al Objeto, la fase de análisis del problema, también llamada inicio, estudia solamente los requisitos de un solo problema. En cambio, en el desarrollo del framework captura los requisitos para un dominio entero. Además, el resultado final del desarrollo orientado al objeto tradicional es una aplicación que es completamente ejecutable, mientras que muchas aplicaciones resultan a partir de la fase de "instanciación" del desarrollo del framework. La fase del "instanciación" abarca las fases de construcción y de transición del desarrollo tradicional. Así, la construcción y las fases separadas de la transición están presentes en cada uno de las instancias del framework. Para cada una de

las instancias del framework hay un esfuerzo de implementación o puesta en práctica introducido por estas fases.

Aunque el desarrollo del framework promete ser muy eficiente, hay varias asuntos que deben ser discutidos. En las secciones siguientes repasaremos siete puntos que deben considerarse al elegir un modelo de framework, en donde cada uno de estos puntos deben ser analizados cuidadosamente; pues no son ni buenos ni malos, sino que son los puntos precisos a chequear. No obstante, es importante tener en cuenta que el desarrollo del framework orientado al objeto es algo relativamente reciente, así como también, la práctica del Diseño Orientado al Objeto y de su puesta en marcha son procesos nuevos de desarrollo. Creemos que el framework se desarrollará y probará la regla del pulgar para muchos dominios, pero seguramente no para todos.

La evaluación de la tecnología del framework presentada aquí se basa en las observaciones recopiladas por los autores acerca del desarrollo y la "instanciación" de varios frameworks para el área del comercio electrónico en nuestro laboratorio, el [TecComm/LES](#). Animamos al lector a que pruebe un framework de comercio electrónico llamado V-Market [14]

Desarrollo del generador de Aplicaciones versus Desarrollo de Aplicaciones

Como habíamos indicado antes, los frameworks generan aplicaciones no por defecto sino que personalizando los requisitos particulares. Ellos en sí mismas no son aplicaciones sino que son construcciones más complejas. Es importante tener presente que el desarrollo de un framework será por lo menos tan costoso como el solo desarrollo de la aplicación. Uno debe analizar cuidadosamente la necesidad de flexibilizar un framework al evaluar los requisitos que se deben resolver para un cliente o futuro usuario, de lo contrario un marmotreto no reutilizable será creado innecesariamente

Por otra parte, el esfuerzo de construir Generadores de Aplicación puede tener éxito a través de la generación repetida de aplicaciones dentro del dominio propuesto. Cuando se elige un buen modelo de framework uno debe preguntarse: "Crearé yo aplicaciones de este mismo dominio más de una vez?" Si la respuesta es sí, entonces es importante evaluar si el trabajo de crear más de una aplicación tendrá éxito al crear un Generador de Aplicación. En suma, *esté atento de los costos versus las ventajas de elegir desarrollar un framework en vez de un sistema de software por encargo.*

Considere el diseño de un sistema para transformar ficheros del texto a partir del uno codificado, por ejemplo el juego de caracteres latinos ISO-8859-1, a una codificación alterna, tal como la codificación del formato del texto del E-mail (MIME, Multi-purpose Internet Mail Extension) [12]. Debiera este sistema ser construido como un framework? La respuesta es sí y si hay planes para convertir el ISO-8859-1 en otros formatos (ej: UUENCODE), o convertir el ASCII en MIME, UUENCODE, u otros posibles formatos futuros. En el primer caso, el hot-spot o punto caliente sería el tipo del texto de la salida. En el segundo caso, el tipo del texto de entrada de información también sería un punto caliente. Por ejemplo, en el cuadro 4 un texto escrito usando ISO-8859-1 tiene caracteres tales como "ñ", "ç" o las vocales acentuadas "á é í ó ú", que no existen en el ASCII, y son codificados en formato MIME y UUENCODE

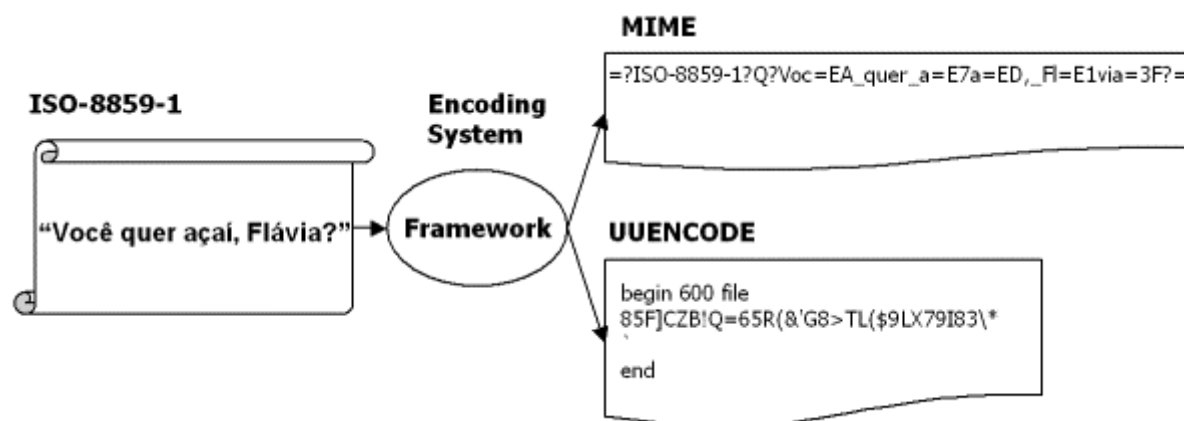


Figura 4. Un ejemplo que muestra un sistema de codificación en el trabajo

Pero qué pasa si este sistema solo convierte textos del ASCII al MIME, pero sin que haya algún plan de desarrollo futuro? En este caso, un framework pudo ser una elaborada aproximación al problema. Lamentablemente, la mayoría de las veces esta opción no está tan clara. De vez en cuando usted mismo podría encontrarlos partes grises o poco claras más a menudo de lo que usted quisiera. Es siempre una buena idea controlar cómo los sistemas similares han resuelto los requerimientos del cliente. Usted puede que incluso descubra que cada sistema similar sería un ejemplo de su framework, pero aún así vale la pena construirlo?

Desarrollo de la Composición

Es común integrar frameworks para satisfacer requerimientos de una aplicación. Sin embargo, Michael Mattson discute que haya por lo menos seis problemas comunes que los desarrolladores de frameworks y aplicaciones encuentran al integrar dos o más frameworks [9]. Todos estos problemas derivan de un conjunto de cinco causas comunes: comportamiento cohesivo, cobertura del dominio, intención del diseño, carencia del acceso al código de fuente, y carencia de los estándares para el framework. Estos problemas se detallan detenidamente [9], donde se proponen muchas soluciones para cada uno.

Si uno desarrolla un framework y espera que sea utilizado, la integración del framework es una realidad inevitable. El desarrollo de la composición no debe ser tomado tan ligeramente. Los frameworks se abandonan o se abortan a menudo porque no pueden ser fácilmente integrados con otros frameworks. La integración del framework no es una tarea fácil, y *la composición debe ser considerada seriamente durante el desarrollo*.

Una forma de considerar la composición cuando se desarrollan frameworks es mantener un conjunto de APIs (API, Application Program Interface, Interface de los programas de aplicaciones), que encapsulan los servicios que el framework proporciona. Al componer con un framework, la aplicación solamente necesita saber algunas funciones y parámetros que debenser llamados, ignorando los funcionamientos internos del framework. Otra opción es crear una capa de mediación, para convertir las peticiones del framework. Sin embargo, si se están componiendo implicando a muchos frameworks, esta aproximación comprueba lo costoso que es esto si uno no elige una capa de mediación unificada entre *todos los* elementos compuestos. En este caso, la capa de mediación actuará como "pegamento" entre estos elementos compuestos.

La Instantiación y la Documentación del Framework

Un framework se puede también clasificar según su extensibilidad; puede ser utilizado como una caja blanca o caja negra [4]. En los **frameworks de caja blanca** también llamados frameworks con arquitectura de configuración-conducidos, la instantiaciones solamente posible a través de la creación de nuevas clases. Estas clases y el código se pueden introducir en el marco por herencia o composición. Uno debe programar el framework y entenderlo muy bien para producir un caso.

Los frameworks de caja negra producen instancias usando escrituras o scripts de configuración. Después de la configuración, una herramienta automática de instantiación crea las clases y el código de fuente (Source Code). Por ejemplo, es posible utilizar un wizard o configurador gráfico que dirija al usuario gradualmente paso por paso, a través del proceso de instantiación del framework. La caja negra del framework no requiere que el usuario sepa los detalles internos del framework. Por lo tanto, estos frameworks también se llaman frameworks dato-conducidos [4] y son generalmente más fáciles de usar. El cuadro 5 ilustra conceptualmente los frameworks de caja blanca y los frameworks de caja negra. Los frameworks que contienen características de ambas cajas se llaman **los frameworks de caja gris**.

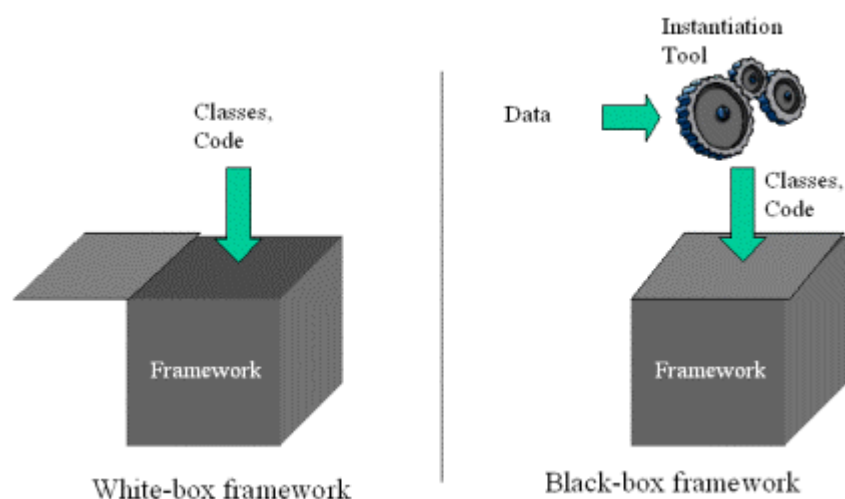


Figura 5. Frameworks de caja blanca vs. Frameworks de caja negra

La caja blanca, la caja negra y la mezclada caja gris tienen una curva de aprendizaje escarpada. Así, la capacidad de crear sistemas ejecutables desde un framework dependerá de la utilidad de los mecanismos de instantiación y de la documentación del framework. Si un framework está pobremente documentada, pocas personas la utilizarán. Sin embargo, un nuevo tipo de documentación debe estar presente con los frameworks: Guías de "Cómo ampliarse" y/o herramientas de instantiación.

Muchas guías de consultas y documentación han sido hechos para los framework. **Las tarjetas de los puntos calientes** [13] se enfoca en los puntos flexibles del framework. Otro enfoque es crear **libros de recetas o cookbooks** que explica cómo el framework debe ser puesto en ejecución y los pasos requeridos para hacerlo [813]. Éstos cookbooks contienen muchas "recetas" que describen de manera amena e informal cómo solucionar problemas específicos mientras usted está instantiando el framework. Un tercer enfoque es asociarlas soluciones arquitectónicas usadas a través del diseño del framework. Sin embargo, la documentación de la configuración del framework no explica todas las facetas del framework.

Según lo dicho antes, las aplicaciones del desarrollo del framework, con frecuencia, usan modelos de diseño. Aunque estos fragmentos de la arquitectura de la configuración constituyen una visión limitada de un framework, estos patrones o modelos son bien conocidos que pueden ayudar a la comprensión del proceso de instantiación del framework. Considere otra vez el ejemplo mostrado en el cuadro 4. En este problema del dominio, uno desea convertir caracteres entre los formatos usando diversos algoritmos (ej: el ISO-8849-1 al algoritmo de conversión MIME). Una solución eficaz puede crear un punto de extensión (punto caliente) que

permite que el algoritmo de la conversión sea alterado de tal manera como si fuese un "plug and play". La *estrategia* de los patrones de diseño satisface a esta aplicación y permite que diversos algoritmos de conversión estén "plugged-in" o enchufados al framework sin alterar el código previamente escrito [5]. Este ejemplo se modela en UML en el cuadro 6. Este diagrama utiliza la misma notación usada en el cuadro 2, con algunos elementos más. El caja de esquina plegable es un comentario, y la línea discontinua indica qué elemento del diagrama se refiere el comentario. Es importante notar que el cuadro 6 puede también servir como parte de la documentación del diseño del framework.

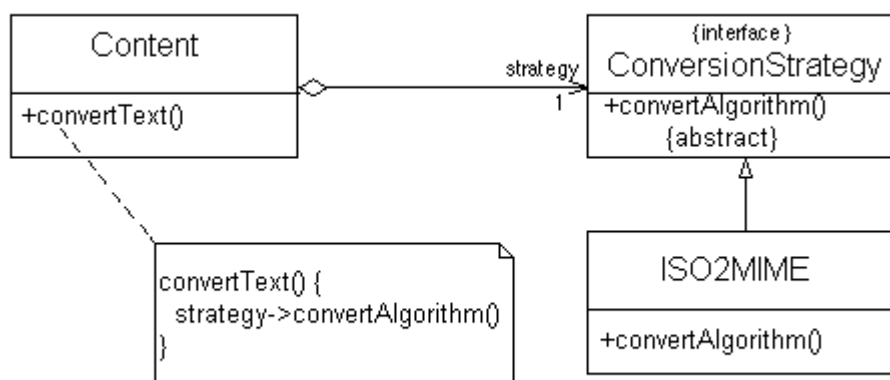


Figure 6. . Un ejemplo de una solución arquitectónica

Aunque hay muchos enfoques posibles y factibles de documentar en un framework, no hay aún un estándar claro. La aproximación más segura es utilizar dos o más enfoques discutidos anteriormente.

El Coste del Análisis del Dominio y la Experiencia

Como se mostró anteriormente, los frameworks se crearon para generar aplicaciones para un dominio específico. Para este propósito, una de las fases del desarrollo del frameworks es el análisis del dominio. A diferencia de la fase del requisito de los sistemas de software, el análisis del dominio cubre una clase entera de problemas

El análisis del dominio intenta caracterizar el tamaño y la complejidad de un dominio elegido. Si el dominio es demasiado grande, éste desperdicia tiempo en recolectar y evaluar la información y sus recursos. Además, el tiempo que se demora en desarrollar el framework y el costo del mismo serían excesivos; es más, será necesario tener individuos familiarizados con el dominio, con los prototipos o los sistemas de software similares. Pero encontrar la experiencia que cubra un dominio grande es bastante difícil.

Por otra parte, si uno elige un dominio demasiado estrecho, se reduce la aplicabilidad del framework y las aplicaciones generadas serán demasiado similares para justificar el esfuerzo de construir un framework. Es importante tener en cuenta qué puntos calientes son necesarios, cuales son los que realmente se necesitan en los requerimientos y los que son innecesarios o superfluos.

Con el tiempo y un framework llega a ser más maduro, cambia y se desarrolla por consiguiente. Este proceso puede representar la alteración de su arquitectura de configuración, pues aparecen nuevos requerimientos que no tenían soporte, y muchas otras nuevas causales. Mientras tanto, las aplicaciones generadas usando el framework también evolucionan y cambian.

Cómo es posible compatibilizar ambos aspectos, el de las aplicaciones y el de la evolución del framework? Las aplicaciones basadas en frameworks pudieron ser dejadas de lado si se cambia o discontinúa un framework. Sin embargo no hay una solución clara, y ciertamente la mayoría de la gente sufrirá con la inestabilidad diciendo que "aquí no pasaría nada," al rechazar utilizar cualquier framework no construido por sí mismos. El único consejo posible para esta situación es estudiar cuidadosamente el framework que se utilizará; un framework bien diseñado e implementado no será tan volátil como mal concebido.

Cuando hablamos que un framework está "bien diseñado" significa que los frameworks deben tener documentación sólida y resolver los requerimientos del dominio. Un framework que mantiene el concepto de encapsulación orientado al objeto y contiene un interfaz público bien definido es probable siga siendo compatible en el nivel de interfaz a través de actualizaciones y revisiones. La evaluación del diseño del framework y/o su implementación no es siempre directa; uno debe considerar la experiencia del diseñador, la complejidad del proceso del instantiación, los requisitos resueltos y no resueltos, y también la actualización del mapa de la ruta de trabajo, que contiene los planes para poner al día el framework, si es que representará un rediseño completo en cada nueva versión o simplemente una subsiguiente actualización compatible.

Flexibilidad vs. Complejidad y Funcionamiento

Como indicamos arriba, los frameworks se construyen para dar flexibilidad y generalidad, tratando de cubrir un dominio entero en vez de problemas determinados. Este enfoque produce un Generador de Aplicación más complejo y más extensible (los puntos calientes) que en los sistemas de software tradicionales. Se alcanza la extensibilidad usando la herencia y la encuadernación dinámica, características comunes en lenguajes orientados al objeto tales como C++, Java o SmallTalk. Por esta razón, un equilibrio entre la flexibilidad y el funcionamiento debe estar presente, puesto que el encuadernamiento dinámico introduce un gasto general, y su uso a través del sistema provocará retrasos e impedimentos en su funcionamiento.

Este equilibrio se hace necesario para el diseñador del framework para elegir los puntos calientes cuidadosamente, sin crear ni generar un framework que sea demasiado genérico. Aunque la flexibilidad es importante y útil, debe estar presente solamente cuando sea necesaria. Si no uno podía idear "una solución universal determinista del framework" ilustrado en el cuadro 7. En este increíble framework, los puntos calientes son los métodos del `problem_not_solved()`, y del `try_to_solve_problem()` y del `return_solution()`. Puede ser instantiados para encontrar la solución de cualquier problema en los problemas deterministas del dominio, pero por supuesto, es inútil.

```
while ( problem_not_solved() == true )  
{  
    try_to_solve_problem();  
}  
return_solution();
```

Figura 7. Solución Universal determinista del framework

Junto con los desarrollos del funcionamiento, el uso abusivo de puntos calientes en un diseño del framework conducirá inevitable a sistemas de software complejos. Usar puntos calientes para introducir soluciones genéricas agrega complejidad al marco. Pues no hay requisitos para los puntos calientes "adicionales", la complejidad agregada no agregará nada en términos de las funciones. Es importante notar que es común que los desarrolladores introduzcan puntos calientes incorrectos pensando que harán un framework "de alcance más grande." Sin embargo, esta aproximación conducirá a la complejidad y los desarrollos del funcionamiento, según lo mostrado arriba, y a veces las funciones adicionales pudieron ser una adición inconveniente.

Problemas con casos de poner a punto el framework.

Los frameworks generan las aplicaciones que tienen entrelazado el código específico de la aplicación y el código congelado del punto. Por lo tanto, un rastro que queda al poner a punto el código de la aplicación conduce a menudo al código del framework. Usar los métodos de pasos simples para poner a punto, no trabajará debidamente a esta mezcla. Las llamadas de puntos congelados se deben separados llamadas del punto caliente.

La distinción automática entre el punto del código congelado y el código del punto caliente es imposible para cualquier depurador. Una solución posible es el uso de pre y poste-condiciones en cada método del punto del código congelado, de esta manera sirviendo como aseveraciones ejecutables asegurándose de que estas llamadas son válidas. Esta manera, las asunción alertará para cualquier posible entrada corrupta de información o feedback corrompida dado al código congelado del punto, y el rastreador sera mucho más fácil. Sin embargo, esta solución no se ocupará de la complejidad introducida por la framework del puntocaliente y del código congelado del punto. Enredando puntos calientes y congelados, el marco será más duro al instantiar, pues los desarrodores de aplicación tendrán que cambiar e introducir un nuevo código cuidadosamente, para evitar de corromper el código congelado del punto que no debe ser alterado. El coste de maintain un código confuso suele ser inevitablemente alto.

Conclusiones

Un enfoque a los frameworks deben considerar cuando los requisitos del producto cambian rápidamente. Los frameworks se pueden también utilizar para aumentar el desarrollo; implementando al principio un códioghot-spot simple y posteriormente actualizarlo. Una buena referencia de frameworks es [4.], aquella que hace una buena evaluación de su metodología y sus últimos avances.

El tema de los frameworks es algo de reciente desarrollo e investigación. En consecuencia, aún quedan muchos problemas por resolver, tales como la documentación del framework. Al cierre de la publicación de este artículo, aún no hay un estándar oficial o de facto para la documentación de los frameworks. La falta de un terreno común crea un espacio vacío entre los desarrolladores de frameworks, los extendedores y los usuarios. La parte económicaes es otro de los problemas en la implementación de los frameworks el cual hay que estimar el costo de construir un framework versus las aplicaciones,y por consiguiente las utilidades que generaría esa inversión. Aún hay pocos ejemplos de framework Industrial (aquellos que estásiendo usados en ambientes comerciales e industriales). Aunque la aplicaciónde la metodología del framework en este contexto es promisorio ,aún está por evaluarse esos esfuerzos, que por ahora son incipientes.[10].

Finalmente creemos que los recién llegados al escenario de desarrollo del framework, debieran chequear los puntos expuestos aquí con sumo cuidado. Considerando lo Ud. necesita, que es lo que está haciendo, y estar conciente de que los frameworks tienen sus pros y sus contras, debido a que no son la solución para todos los problemas. Además, si Ud. está considerando usar un framework ya existente, estudie su documentación y verifique si hay una buena explicacióna cerca de "Cómo Instantiar" , si es que existe. Es importante observar también, las aplicaciones que fueron generados y la cantidad desfuerzo dedicado a este proceso.

Referencias

1

Booch, G., Jacobson, I., Rumbaugh, J., and Rumbaugh, J. *The Unified Modeling Language User Guide*. Addison-Wesley Pub Co, 1998.

- 2 Brooks, Jr., F. P. *No Silver Bullet--Essence and Accidents of Software Engineering*. IEEE
Computer 20(4), April, 10-19, 1987.
- 3 --, *CORBA Website*. url: <http://www.corba.org/>.
- 4 Fayad, M. E., Schmidt, D. C., and Johnson, R. E. *Building Application Frameworks*.
Addison-Wesley Pub Co, 1st edition, 1999.
- 5 Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns : Elements of
Reusable Object-Oriented Software*. Addison-Wesley Pub Co, 1st edition, January 1995.
- 6 Jacobson, I., Booch, G., Rumbaugh, J. *The Unified Software Development Process*.
Addison-Wesley, 1999.
- 7 --, *JUnit, Testing Resources for Extreme Programming*, url: <http://www.junit.org>.
- 8 Mattsson, M. *Object-Oriented Frameworks: A Survey of Methodological Issues*. Technical
Report 96-167, Dept. of Software Eng. and Computer Science, University of
Karlskrona/Ronneby.
- 9 Mattsson, M., Bosch, J., and Fayad, M.E. *Framework Integration Problems, Causes,
Solutions*. Communication of the ACM October 1999/Vol.42, No.10.
- 10 Mattsson, M. and Bosch, J. *Observations on the Evolution of an Industrial OO
Framework*. Proceedings of ICSM'99, International Conference on Software Maintenance,
Oxford, UK, 1999.
- 11 --, *Microsoft COM Technologies*. url: <http://www.microsoft.com/com/>.
- 12 --, *MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header
Extensions for Non-ASCII Text*. Request for Comments (RFC) 2047, url: [http://www.rfc-
editor.org/rfc/rfc2047.txt](http://www.rfc-editor.org/rfc/rfc2047.txt).
- 13 Pree, W. *Design Patterns for Object-Oriented Software Development*. Addison-Wesley
Pub Co, March 1995.
- 14 Ripper, P., Fontoura, M. F., Neto, A. M., and Lucena, C. J. *V-Market: A Framework for e-
Commerce Agent Systems*. World Wide Web, Baltzer Science Publishers, 3(1), 2000.
-

Biografía

Marcus Eduardo Markiewicz posee un Master en Ciencias de la Computación, como estudiante graduado de la Pontificia Universidad Católica de Rio de Janeiro, Brasil (PUC-Rio). Se dedica completamente a la investigación en el [grupoTecComm e-Commerce](#) del [Laboratoriode Ingeniería del Software \(LES\)](#) en la [PUC-Rio](#).

Carlos J.P. Lucena es profesor universitario del [Departmentode Ciencias de la Computación](#) en la [PUC-Rio](#) desde 1982. Recibió en 1965 su grado de Licenciado en la Pontificia Universidad Católica de Rio de Janeiro (PUC-Rio), Brasil. En 1969 obtuvo su Maestría en Matemáticas en Ciencias de la Computación en la [Universidad deWaterloo](#), Canadá. En 1974 obtuvo su Doctorado en Ciencias de la Computación en la [Universidadde California en Los Angeles](#) . Además es miembro del Comité Editorial de la Jornada Internacional sobre Aspectos Formales de la Computación, en inglés: Editorial Board of International Journal on Formal Aspects of Computing (New York: Springer-Verlag).

Location: <http://www.acm.org/crossroads/espanol/xrds7-4/frameworks.html>

