UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 4

Título: Propuesta de Procedimiento de Evaluación de los Frameworks. Un enfoque práctico.

Trabajo de Diploma para optar por el título de
Ingeniero en Ciencias Informáticas.

Autor: Livan Kabir Badías Ibarra

Tutor: M. Sc. Eduardo Escofet Batista

Ciudad de la Habana, julio 2007.
"Año 49 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Livan Kabir Badías Ibarra	M. Sc. Eduardo Escofet Batista
_____	_____
Firma del Autor	Firma del Tutor

DATOS DE CONTACTO

Datos del Tutor

Nombre: Eduardo Luis Escofet Batista **Edad**: 36 años

Email: escofet@facinf.uho.edu.cu

Datos Profesionales

- Graduado de Licenciatura en Cibernética-Matemática, Universidad Central de Las Villas, Cuba, 1994.
- Máster en Informática para la Administración, Universidad de Holguín, Cuba, 1997.
- Diploma de Estudios Avanzados en Informática Universidad de Granada, España, 2004.
- Desde hace 12 años trabaja en la Universidad de Holguín, Cuba. Responsable Grupo Investigación de Sistemas Distribuidos (GISIDI).

Principales postgrados recibidos:

- Curso de Redes Neuronales Artificiales aplicadas (En Universidad de Cádiz. España, 1997).
- Curso de Programación Java (Prof. Mario Méndez Lojo, España, 1998).
- Curso de Doctorado en Integración Empresarial (Dr. Ricardo Chalmeta Jefe del Grupo IRIS de la Universidad Jaume I (UJI), España, 2000).
- Curso de Doctorado en Reingeniería de Sistemas (Dr. Ricardo Chalmeta, UJI, España, 2000).
- Curso de Doctorado de Interfaces Avanzadas de Sistemas Multimedia (Dr. Pedro J. Sanz, UJI, España, 2000).
- Curso de Doctorado Computación de Altas Prestaciones (Dr. José Manuel Badía, UJI, España, 2001).
- Curso de Doctorado Programación Paralela de Multicomputadores (Dr. José Manuel Badía, UJI, España, 2001).
- Curso de Doctorado Nuevos Paradigmas de la Programación (Dra. María José Fórtiz, Universidad de Granada, España, 2003).

- Curso de Sistemas Gráficos y Multimedia (Dr. Juan Carlos Torres, UGR, España, 2003).
- Curso de Arquitectura de Computadores (Dr. Julio Ortega, UGR, España, 2003).
- Curso de Reconocimiento de Patrones (Dr. Ignacio Requena, UGR, España, 2003).
- Curso de Soft-Computing (Dr. José Luis Verdegay, UGR, España, 2003).
- Curso de Lógica Difusa y BBDD Difusas (Dra. Amparo Vila, UGR, España, 2003).
- Curso de Sistemas Multiagentes (Dr. Miguel Delgado, UGR, España, 2003).
- Talleres sobre Sistemas Colaborativos y Evolución del Software en la Universidad de Granada, España, 2004.
- Curso de Programación Web Avanzada, UHO 2007.

Principales postgrados impartidos:

- Ingeniería del Software para Maestría de Informática para la Administración, UHO, 1997.
- Curso de diseño de aplicaciones Web para empresas turísticas de la región, Hotel Meliá Río de Oro.
- Redes de computadoras e Internet para Maestría de CAD/CAM/CAE, UHO, 1999.
- Cursos de Programación, Estructura de Datos y Análisis de Algoritmos a la carrera de Ingeniería Informática.
- Cursos de Introducción a la Informática y Bases de Datos en idioma Inglés. Universidad de Belice.
- Introducción al Lenguaje C#, UHO, 2003.
- Programación Orientada a Objetos con Java, UHO, 2005.
- Herramientas de Programación Avanzada, UHO 2006-2007.
- Tecnologías de la Informática, UHO 2006-2007.
- Desarrollo de sistemas Informativos, UHO 2006-2007.

AGRADECIMIENTOS

En primer lugar gracias a toda mi familia por su generosidad y ayuda.

Mi agradecimiento más sincero para Escofet, por tu valiosa dedicación, paciencia y buen humor. Gracias a ti empezó y se ha llevado a cabo este trabajo.

También quiero agradecer su colaboración y ánimos a todos mis amigos y compañeros de estudios.

Mi eterno agradecimiento y gratitud a nuestra Revolución y nuestro Comandante Fidel por darme la posibilidad de realizar mis sueños.

DEDICATORIA

Dedicado a mi familia. A Mildre, Jesús, Liuba, Midelkis,

y especialmente a mi niña Nathaly.

OPINIÓN DEL TUTOR

Título del trabajo de diploma: PROPUESTA DE PROCEDIMIENTO DE EVALUACIÓN DE LOS FRAMEWORKS. UN ENFOQUE PRÁCTICO.

Tutor del trabajo de diploma: M. Sc. Eduardo Luis Escofet Batista

La calidad de la investigación y el desarrollo del diplomante ha sido excelente a lo largo de todo el trabajo, mostrando independencia, responsabilidad y alta creatividad al desarrollar ideas y herramientas no existentes en el estado del arte actual. La bibliografía actualizada en su gran parte pertenece a los años 2004, 2005 y 2006, casi toda en Inglés, demostrando un dominio de las técnicas de búsqueda y condensación de alta calidad en Inglés y Español. Ha abarcado el contenido del trabajo asignado, llegando a casi todas las áreas de conocimiento del Ingeniero Informático en el tiempo de I+D de su tesis de diploma. Se destaca la profundidad de esta investigación no común en tesis de grado, así como la efectividad de las tecnologías y herramientas generadas. El procedimiento presentado ha sido valorado y aplicado en la práctica con el framework *freeTribe* para el desarrollo de sistemas colaborativos distribuidos, obteniéndose resultados importantes en su rediseño y eficiencia.

Considero que el estudiante se encuentra listo para ejercer como Ingeniero en Ciencias Informáticas, y se le propone la calificación de __5__ puntos Excelente.

Para que así conste se expide la presente a los ___ días del mes de _____ del año _____.

_____

Tutor: M. Sc. Eduardo Escofet Batista

Prof. del Dpto. Informática

J' Disciplina de Programación

Universidad de Holguín

RESUMEN

En atención a las múltiples dificultades que enfrentan los desarrolladores de software para construir frameworks de calidad, fáciles de mantener y evolucionar, se efectuó la investigación que se expone a continuación, con el objetivo de diseñar un procedimiento que expone y evalúa las principales propiedades y/o principios que caracterizan a los frameworks, aplicable durante el proceso de ingeniería del framework así como después de realizado, como una herramienta para su evolución y/o certificación de parámetros de calidad.

Para la consecución del objetivo propuesto se realizó una amplia revisión bibliográfica, lo que permitió elaborar y exponer la fundamentación teórica de los frameworks, así como del proceso para su desarrollo, haciendo énfasis en las limitaciones que condujeron al desarrollo del presente trabajo.

Así mismo, se presentó el procedimiento propuesto para la solución del problema, su definición y esquema general; el cual, para una mejor comprensión, se aborda en dos fases generales: definición del marco conceptual y, evaluación.

El procedimiento propuesto tiene un carácter iterativo, lo que permite la inclusión de nuevas propiedades y facilita la rectificación de defectos u omisiones encontrados, facilidad que se garantiza por medio de estrategias expuestas durante las iteraciones de la fase de evaluación.

Con el propósito de obtener una valoración preliminar que permitiera, además, la mejora del procedimiento, incluso antes de su salida, se realizaron encuestas a expertos en el tema y, se aplicó en la práctica con el framework *freeTribe* para el desarrollo de sistemas colaborativos distribuidos, obteniéndose resultados importantes para su rediseño y eficiencia.

Palabras claves: frameworks, arquitectura, propiedades y principios de los frameworks, evaluación.

TABLAS Y FIGURAS

ÍNDICE

INTRODUCCIÓN

Los seres humanos somos “imperfectos” y, como dijera Fidel Castro Ruz, “…ninguna obra humana es perfecta”. El hombre desarrolla software dirigido a facilitar, dinamizar y humanizar el trabajo; en cada software se pretende la solución más acabada, la ideal pero nunca se alcanza la perfección.

Por otra parte, las exigencias para la solución de problemas cada vez más complejos, resultan muy rigurosas e introducen premisas ineludibles al desarrollo de software:

- **Complejidad**: los sistemas cada vez son más y más complejos, lo que presupone que “a más líneas de código, más errores”.
- **Extensibilidad**: los sistemas deben permitir incorporar funcionalidades sin necesidad de recompilar la totalidad del código.
- **Conectividad**: los sistemas deben estar preparados para interactuar con otros sistemas.
- **Reusabilidad**: los sistemas deben la posibilidad de reusar software de otros proyectos o desde partes del mismo proyecto.

Evidentemente la construcción de aplicaciones software es cada vez más compleja, ello impone la necesidad de solución que permitiera:

- Reusar el diseño y el código fuente.
- Aumentar la productividad de los desarrolladores: generación de código, etc.
- Flexibilidad para cambiar los requerimientos del negocio así como el panorama tecnológico.

Como las personas, las aplicaciones de software tienen más aspectos en común que diferentes. Por ejemplo: se ejecutan en las mismas computadoras, esperan por entrada de datos desde los mismos dispositivos, muestran los resultados a través de los mismos dispositivos, salvan datos en los mismos medios de almacenamiento, etc. Una de las soluciones (no quiere decir que sea la única) a estas problemáticas lo constituyen los “frameworks”. Un framework es una aplicación reusable y semicompleta

que puede ser especializada para producir aplicaciones personalizadas o específicas (**Johnson, 88**). En contraste con las prematuras técnicas de reuso orientadas a objetos basadas en librerías de clases, los frameworks apuntan a unidades de negocio específicas, tales como: procesamiento de datos, comunicaciones, etc., y a dominios de aplicaciones, como son: interfaz gráfica de usuario, persistencia, etc.

Existen varios principios y/o propiedades que caracterizan y aportan solidez a los frameworks, entre los que se destacan: inversión de control, separación de intereses, extensibilidad, etc. y, por lo tanto, deben ser evaluados durante o después de la construcción del framework (en las primeras fases del proceso de desarrollo sería más rentable).

En el trabajo que se presenta se tendrá en cuenta el siguiente **problema científico**: la no existencia de un procedimiento eficiente para la evaluación y caracterización de las plataformas de desarrollo, específicamente los frameworks.

El **objeto de estudio** en el cual se enmarca el problema planteado anteriormente es: la arquitectura de frameworks.

Y el **campo de acción**: propiedades y/o principios esenciales de los frameworks de desarrollo.

Para dar solución al problema científico, el trabajo se propone el siguiente **objetivo**: diseñar un procedimiento que exponga y evalúe las principales propiedades y/o principios que caracterizan a los frameworks de desarrollo, aplicable durante el proceso ingeniería del framework así como después de realizado como una herramienta para su evolución y/o certificación de parámetros de calidad.

Para el diseño de la investigación se plantea la siguiente **hipótesis**: Un procedimiento que proporcione una guía para la determinación, caracterización y valoración de las propiedades fundamentales de un framework, facilitará su construcción, evolución y documentación satisfactoria, al revelar defectos u omisiones que atenten contra la calidad y solidez del mismo.

Para el desarrollo de esta investigación se realizaron las siguientes **tareas investigativas**:

1. Fundamentación teórica de los frameworks, y del proceso para su desarrollo.
2. Presentación de un conjunto no exhaustivo de las propiedades o principios que caracterizan a los frameworks.
3. Proceso de determinación, caracterización y valoración de estas propiedades.
4. Aplicación del proceso al framework *freeTribe* **(Rodríguez, 06)**.

Para darle cumplimiento a las tareas antes mencionadas se emplearon los siguientes métodos de investigación:

Métodos teóricos:

- **Análisis y síntesis**: Para procesar la información en la elaboración de los fundamentos teóricos y de las conclusiones.
- **Histórico lógico**: Para el estudio de la evolución del problema y de la existencia de modelos similares al que se pretende elaborar. De existir alguno, conocer los resultados alcanzados tras su aplicación.
- **Hipotético deductivo**: Para la elaboración de la hipótesis y la deducción de los resultados de la investigación.

Métodos empíricos:

- **Encuestas**: Para estudiar el estado de opinión de los especialistas sobre el proceso de evaluación de los frameworks, así como las posibilidades de uso del procedimiento propuesto.

CAPÍTULO 1

ESTUDIO DEL CONTEXTO DE LA INVESTIGACIÓN

1.1 Introducción

Los frameworks son de importancia capital para construir grandes sistemas de software orientado a objetos y ofrecen, a través del reuso tanto del diseño como de las implementaciones, una mayor productividad y una disminución considerable del tiempo de salida al mercado del producto, a través del reuso tanto del diseño como de las implementaciones. Sin embargo, esta premisa es difícil de cumplir a cabalidad.

En este capítulo se proporciona una visión general de los principales conceptos relacionados a los frameworks y al proceso para su desarrollo, para ello se ha realizado un estudio profundo del estado del arte del tema en cuestión, haciendo énfasis en las falencias que han conducido al desarrollo del presente trabajo.

1.2 Fundamentos teóricos de los frameworks

El tema de los frameworks es algo de reciente desarrollo e investigación. En el presente capítulo se exponen los conceptos fundamentales relacionados a los frameworks, así como el proceso de ingeniería para el desarrollo de los mismos.

1.2.1 Definiciones de framework

Muchos de los que se dedican al desarrollo de software utilizan, conocen o, como mínimo, se han tropezado con el concepto de framework, cuya traducción aproximada sería “marco de trabajo”. Sin embargo, el concepto de framework no es tan simple y mucho menos sencillo de definir; aunque cualquiera con experiencia en la programación captará su sentido de manera casi intuitiva e incluso es posible que esté utilizando su propio framework. Resulta pertinente citar las siguientes definiciones:

- “…diseño abstracto orientado a objetos para un determinado tipo de aplicación, que se compone de una clase abstracta para cada componente principal del diseño; contendrá normalmente una librería de subclases que pueden ser utilizadas como componentes del diseño…” (**Johnson, 88**).

- "…patrón arquitectónico que proporciona una plantilla extensible para aplicaciones dentro de un dominio…" (**Booch, 99**).

La palabra framework recoge en su definición conceptos familiares, tales como "clase abstracta", "interfaz", "patrón" y "extensibilidad". Además un framework está asociado a un determinado tipo de aplicaciones, lo que implica que su alcance esté acotado; no es un diseño de propósito general, sino que está ligado a un dominio concreto. En ese sentido, se usa también la terminología "plantilla extensible" para dar a entender, por un lado, que se compone de una estructura fija de componentes con una determinada funcionalidad, y por otro lado, que es una estructura común para un determinado tipo de aplicaciones que puede ser extendida con nuevos componentes.

El término "patrón arquitectónico" denota la reusabilidad de la arquitectura del diseño (**Jiménez, 03**). En el desarrollo de software la palabra "arquitectura" se considera tanto un sustantivo como un verbo. Como sustantivo, la arquitectura comprende la organización y estructura de los elementos importantes del sistema. Más allá de esta definición estática, incluye el comportamiento del sistema, especialmente en función de responsabilidades de gran escala de los sistemas y subsistemas, y sus colaboraciones. En cuanto a una descripción, la arquitectura comprende las motivaciones o fundamentos de por qué el sistema está diseñado de una determinada manera. Como verbo, la arquitectura es en parte investigación y en parte trabajo de diseño; por claridad, el término es mejor que se califique como en investigación arquitectural o diseño arquitectural (**Larman, 04**).

Un framework incorpora una o varias APIs que describen cómo tener acceso a las diferentes bibliotecas de clases que representan los puntos calientes (hot spots) de dicho framework. A este conjunto de clases relacionadas y reutilizables, diseñadas para proporcionar funcionalidades útiles de propósito general se le conoce como ToolKit **(Gamma, 03**). Los ToolKit no imponen un diseño particular en una aplicación, simplemente proporcionan funcionalidades que pueden ayudar a realizar su trabajo.

Como resumen, se puede decir que un framework (**Larman, 04**):

- Es un conjunto cohesivo de interfaces y clases que colaboran para proporcionar los servicios de la parte central e invariable de un subsistema lógico (frozen spot).
- Contiene clases concretas, y especialmente, abstractas, que definen las interfaces a las que ajustarse, así como interacciones de objetos en las que participar.
- Normalmente requiere que el usuario del framework defina subclases de las clases del framework existentes para utilizar, adaptar y extender los servicios del mismo.
- Tiene clases abstractas que podrían contener tanto métodos abstractos como concretos.
- Confía en el principio de Hollywood: "No nos llame, nosotros le llamaremos", lo que implica que las clases definidas por el usuario recibirán mensajes desde las clases predefinidas por el framework. Estos mensajes normalmente se manejan implementando métodos abstractos en las superclases, esta es la diferencia fundamental entre un framework y una biblioteca de clases (**Booch, 99**).

Una biblioteca de clases contiene abstracciones que las abstracciones del desarrollador pueden invocar o instanciar; un framework contiene abstracciones que pueden invocar o instanciar a las abstracciones del desarrollador. Ambos tipos de conexión constituyen los elementos variables del framework (puntos calientes), que deben ser ajustados para adaptar el framework al contexto específico del problema que se está modelando.

Es importante aclarar que los frameworks no son ejecutables (**Figura I**), para generar un ejecutable se debe instanciar el framework correspondiente al dominio que incluye al ejecutable. Una vez que el framework es instanciado, sus puntos congelados hacen llamadas a las clases definidas por el usuario cuando ocurre un determinado evento. Estos puntos congelados al ser inmutables constituyen el núcleo del framework. El

núcleo será constante y estará presente en todas las aplicaciones que instancien el framework. Por esta razón un framework se identifica a veces como “el viejo código que llama al nuevo código”.

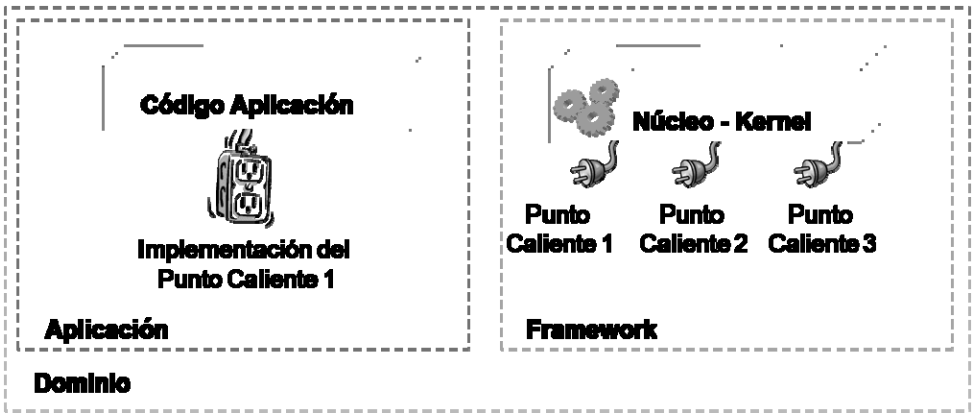

Figura I Instanciación de un framework.

1.2.2 Clasificación de los frameworks

Fayad y Schmidt en (**Fayad, 97**) clasifican los frameworks según cómo éste puede ser extendido para adaptarse o personalizarse a las condiciones específicas del dominio de la aplicación. A tenor con esta clasificación los frameworks pueden ser de caja blanca, de caja negra y, además, existen una gran variedad de híbridos que podemos denominar de caja gris.

En los **frameworks de caja blanca**, también llamados frameworks con arquitectura de configuración-conducidos, la instanciación solamente es posible a través de la creación de nuevas clases. Estas clases pueden instanciar el framework por herencia o composición, lo que implica que el usuario del framework debe conocerlo a plenitud.

Los **frameworks de caja negra** producen instancias usando ficheros o scripts de configuración. Después de la configuración, una herramienta automática de instanciación crea las clases y el código fuente correspondiente. Por ejemplo es posible utilizar un configurador gráfico (wizard) que dirija al usuario del framework gradualmente paso por paso a través del proceso de instanciación, por lo que el usuario no necesita conocer los detalles internos del framework. A estos frameworks

también se les conoce como frameworks dato-conducidos (**Fayad, 99**) y son generalmente más fáciles de usar. En la **Figura II** se ilustran gráficamente los frameworks de caja blanca y los frameworks de caja negra.

Los frameworks que contienen características de ambas cajas, como mencionamos con anterioridad, se llaman frameworks de **caja gris**.

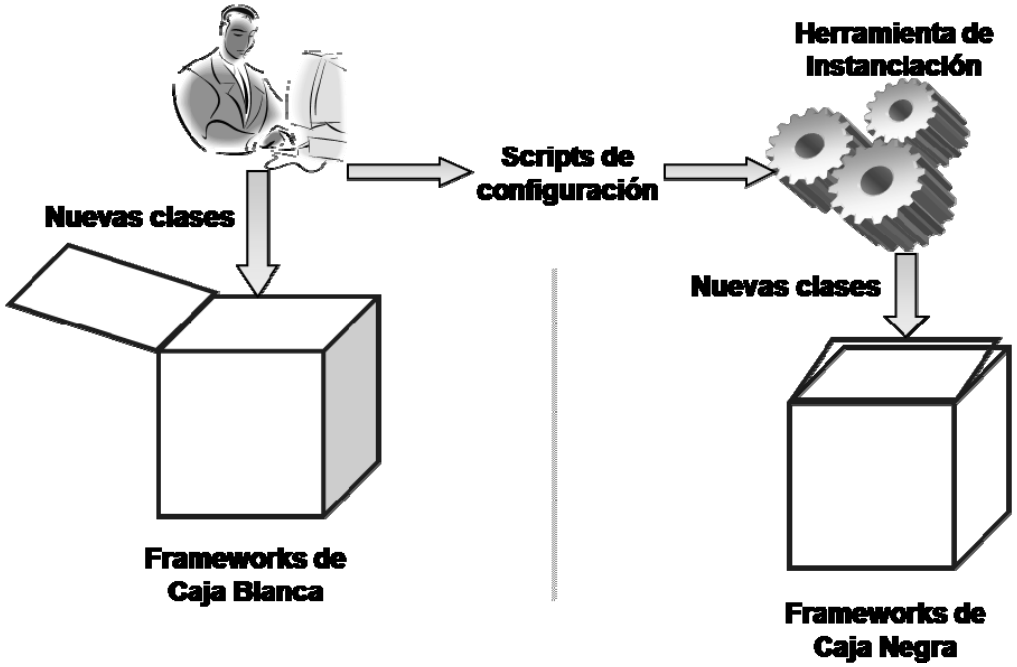

Figura II Clasificación de los frameworks según su extensibilidad.

En (**Taligent, 94**) se da otra clasificación de los frameworks, agrupándolos en tres categorías:

1. **Frameworks de aplicación**: encapsulan una capa de funcionalidad horizontal que se puede aplicar en la construcción de una gran variedad de aplicaciones. Uno de los ejemplos más completos de este tipo de frameworks los constituye los que implementan las interfaces gráficas de usuario (GUI).
2. **Frameworks de soporte**: proporcionan servicios básicos a nivel de sistema.
3. **Frameworks de dominio**: aplicables a un dominio de aplicación o a una línea de productos, tales como aplicaciones bancarias, tráfico, etc. Implementan una capa de funcionalidad vertical. Estos frameworks son y deben ser los más

numerosos, y su evolución deberá ser también la más rápida, pues deben adaptarse a las áreas de negocio para las que fueron diseñados. Este tipo de framework puede hacer uso de las dos categorías mencionadas anteriormente con el objetivo de proporcionar un marco de trabajo completo y robusto para el desarrollo de aplicaciones dentro del dominio para el cual fueron diseñados. Lo que permite a las organizaciones desarrollar software de más calidad y reducir el tiempo de salida al mercado de dicho producto, aumenta la productividad.

1.3 Otras propuestas de reusabilidad

La reutilización no sucede por accidente, necesitamos pensar en ella desde el inicio, pues requiere una posición correcta, herramientas y técnicas adecuadas. Existen varias técnicas para la reutilización del software y los frameworks son una de esas técnicas para reutilizar el diseño, código fuente y otros artefactos creados durante el desarrollo del software. Para lograr su objetivo, los frameworks se conjugan con otras técnicas, tales como patrones, compontes, librerías de clases, etc.

1.3.1 Patrones

En la terminología de objetos, un patrón es una descripción de un problema y su solución, que recibe un nombre y que puede emplearse en otros contextos (**Larman, 99**). La “Banda de los cuatro” (**Gamma, 95**) acuña la frase “patrones de diseño”. Un patrón de diseño nombra, motiva y explica un diseño general que resuelve un problema de diseño que ocurre sistemáticamente en sistemas orientados a objetos. Describe el problema, la solución, cuándo aplicar dicha solución y sus consecuencias; además propicia sugerencias y ejemplos para su implementación. La solución consiste en una disposición general de objetos y clases que resuelven el problema. La solución es personalizada e implementada para resolver el problema dado en un contexto en particular.

Un diseñador que esté familiarizado con los patrones, puede aplicarlos a problemas de diseño ya conocidos, sin tener que redescubrirlos. De esta forma los patrones facilitan la reutilización de diseños y arquitecturas exitosas. Los patrones nos permiten decidir

entre alternativas de diseño que pueden hacer al sistema reusable y evitar las alternativas que comprometan esta característica deseada en todo software.

Ambos, patrones y frameworks, facilitan la reutilización a través de la captura de exitosas estrategias para el desarrollo de software. La principal diferencia es que los frameworks están enfocados en el reuso de diseños, algoritmos e implementaciones concretas en un determinado lenguaje. En contraste, los patrones están centrados en el reuso de diseños abstractos.

En (**Gamma, 95**) se definen las mayores diferencias entre los patrones de diseño y los frameworks:

1. Los patrones de diseño son más abstractos que los frameworks. Los frameworks se pueden contener o envolver en código fuente, pero sólo los ejemplos de los patrones pueden estar contenidos en código. La fortaleza de los frameworks es que éstos pueden ser escritos completamente utilizando algún lenguaje de programación y no sólo estudiados, sino que se pueden ejecutar y reusar directamente. Por el contrario, los patrones de diseño tienen que ser implementados cada vez que se vayan a usar. Los patrones de diseño además explican lo que tratan de hacer, las ventajas y consecuencias de su uso.
2. Los patrones de diseño son elementos arquitectónicos más pequeños que los frameworks. Un framework común contiene varios patrones de diseño, pero lo contrario nunca ocurre.
3. Los patrones de diseño son menos especializados que los frameworks. Se pueden usar en casi cualquier tipo de aplicación. Por muy específico que sea un patrón de diseño, nunca impone una arquitectura a la aplicación.

Los frameworks pueden ser vistos como un conjunto concreto de familias de patrones de diseño que son apuntados hacia un dominio específico de aplicaciones, los patrones proporcionan un nivel de abstracción intermedio entre las aplicaciones y las clases y objetos contenidos en el framework. Asimismo los patrones de diseños pueden ser

vistos como elementos abstractos de la micro-arquitectura de los frameworks, que documentan e incentivan la semántica del framework de forma efectiva.

Cuando los patrones son usados para estructurar y documentar el framework, aproximadamente todas las clases de éste juegan un rol bien definido y colaboran eficientemente entre sí.

1.3.2 Librería de clases

Dos cualidades que pueden distinguir rápidamente una librería de clases de un framework son la integridad y la extensibilidad. Una librería de clases ejecuta una tarea completa como abrir un fichero, calcular el área de un círculo, etc. A diferencia, un framework provee la estructura, pero no una solución completa para una categoría específica de problemas; la solución queda formada cuando el usuario le añade comportamiento al framework.

La extensibilidad es útil a la hora de distinguir entre los frameworks y las librerías de clases. Por ejemplo, podemos tener una solución a un problema determinado, para la persistencia, que se pueda instalar y usar inmediatamente, esta solución puede ser un framework, pues normalmente, debe ser extensible, ya que dicha solución debe permitir que el usuario cree clases personalizadas y modifique las configuraciones para extender su comportamiento, por ejemplo, para persistir objetos en ficheros XML. Las librerías de clases son comúnmente usadas con un comportamiento de **caja negra**, no exponen el funcionamiento de las clases.

En la práctica, los frameworks y las librerías de clases son tecnologías complementarias. Por ejemplo, los frameworks frecuentemente usan librerías de clases para simplificar el desarrollo del propio framework. Asimismo, el código específico de una aplicación que es invocado por los manejadores de eventos del framework puede utilizar las librerías de clases para realizar tareas básicas como el manejo de cadenas, la administración de ficheros, análisis numéricos, etc.

1.3.3 Componentes

Un componente es un bloque de código reutilizable del software: un segmento de código de aplicación encapsulado construido con anterioridad, que puede ser combinado con otros componentes y con código adicional para desarrollar una aplicación determinada. No existe un acuerdo común acerca de qué es y qué no es un componente. Los componentes pueden ser tan simples como un botón en una Interfaz GUI, o tan complejos como un componente para las funciones de administración de redes.

Existen varias definiciones para el término “componente”, de las cuales exponemos a continuación dos:

- “Un componente de software es una unidad de composición con un contrato con las interfaces especificadas y dependencias explícitas del contexto solamente. Un componente de software se puede desplegar independientemente y está conforme a la obra de terceras personas” - **Clemens Szyperski** (**Szyperski, 98**).
- “Un componente es una parte no trivial, casi independiente, y reemplazable de un sistema que satisface una función clara en el contexto de una arquitectura bien definida. Un componente se conforma con un conjunto de interfaces y proporciona la realización o implementación de dichas interfaces” - **Philippe Krutchen** (**Krutchen, 95**).

Dicho de otra manera, en términos de diseño orientado a objetos, los componentes son una caja negra que define un conjunto coherente de operaciones, que pueden ser reusadas basadas exclusivamente en el conocimiento de la sintaxis y la semántica de su interfaz.

Existen una serie de propiedades que deben tener los componentes reusables (**Kaisler, 05**):

- **Aditividad**: capacidad de combinar componentes con efectos secundarios mínimos.
- **Independencia**: cada componente incorpora y desarrolla una sola idea.

- **Simplicidad**: interfaz simple, con un mínimo de parámetros.
- **Modificable**: fácil de modificar, con efectos secundarios mínimos y obvios.

Una motivación importante para usar componentes es la capacidad de reutilizarlos en otras aplicaciones. Durante el diseño de una aplicación, se debe estar pensando acerca de qué componentes pueden ser reusados en otros proyectos similares. Aunque es posible diseñar software con la reutilización como objetivo, se debe ser muy cuidadoso de no diseñar componentes que no puedan ser reusados, esto parece una contradicción, pero no lo es, enfatiza que en cada aplicación, por sus características específicas, existirán componentes que serán útiles única y exclusivamente para el entorno de dicha aplicación. Las soluciones reusables enfatizan en la simplicidad; se puede llegar a la complejidad mediante la reutilización de componentes.

La relación entre frameworks y componentes es altamente cooperativa, sin subordinarse el uno al otro. Los frameworks pueden ser usados para desarrollar componentes, con lo cual la interfaz del componente provee una "fachada" para la estructura interna del framework. Asimismo, los componentes pueden ser usados como estrategia de acoplamiento en los frameworks de caja negra. A menudo los frameworks son usados para simplificar el desarrollo de infraestructuras y aplicaciones intermedias, mientras que los componentes con frecuencia son usados para simplificar el desarrollo de aplicaciones destinadas al usuario.

1.3.4 Contenedores

Según Rod Johnson en (**Johnson, 04**)**,** un contenedor es un espacio en el cual corre el código de la aplicación. Entiéndase código como objetos de la aplicación, generalmente objetos de negocio; éstos corren dentro del contenedor y, usualmente son administrados por éste. Según el propio autor (**Johnson, 04**), un contenedor debe brindar un conjunto de servicios, tales como:

- **Manejo del ciclo de vida**: el contenedor debe controlar el ciclo de vida de los objetos de aplicación que corren dentro de él; maneja la creación, activación y destrucción de dichos objetos, etc.

- **Operaciones de búsqueda**: el contenedor debe proporcionar una vía para obtener referencias a los objetos manejados. Las operaciones de búsqueda son un servicio base.
- **Configuración**: el contenedor debe proporcionar una forma constante de configurar los objetos que funcionan dentro de él, para permitir la parametrización. Los valores simples de la configuración deben ser independientes del código de Java, por ejemplo en ficheros XML, para poderlos cambiar sin la re compilación o la necesidad de implicar a los desarrolladores.
- **Resolución de dependencias**: el contenedor debe ser capaz de manejar las relaciones y dependencias entre los objetos que corren dentro de él.

Los contenedores se pueden definir como ambientes que proporcionan servicios específicos a los componentes y objetos que corren en él.

1.4 Uso de los frameworks

La utilización de un framework en el desarrollo de una aplicación implica un cierto coste inicial de aprendizaje, aunque a largo plazo con seguridad facilitará tanto el desarrollo como el mantenimiento de la aplicación.

Beneficios y desafíos al usar un framework.

Podemos categorizar los beneficios que aporta el uso de los frameworks basado en las siguientes etapas del ciclo de vida del software:

- Diseño.
- Desarrollo y prueba.
- Producción y mantenimiento.

Diseño:

Adoptar un framework dinamiza el proceso de diseño y como los frameworks están alineados con los requisitos del negocio, el diseño se simplifica. Varias de las funcionalidades que tendrían que ser diseñadas, forman parte del framework, como

resultado, el diseño estará centrado en los problemas del negocio y los detalles del mismo.

Desarrollo y prueba:

Los frameworks mejoran el desarrollo y las pruebas proporcionando los siguientes beneficios:

- **Mayor productividad del desarrollador**: al estar los frameworks enmarcados en un dominio, permiten que los desarrolladores sean productivos inmediatamente.
- **Menos codificación**: muchos de los códigos personalizados, ahora son reemplazados en su totalidad por el framework, o es una configuración del mismo; por lo que existe menos código a probar.
- **Mayor consistencia**: con frecuencia se encuentran varias soluciones al mismo problema entre los miembros del equipo de desarrollo; si empleamos un framework, creamos una propuesta consistente para usar a través del desarrollo del software.
- **Resultados inmediatos**: al dinamizar el proceso de asociación de los requerimientos a funcionalidades se reduce considerablemente el tiempo de desarrollo. Esto permite un mayor número de iteraciones en un corto plazo, lo que conlleva a un producto de más calidad y mayor inmediatez en la entrega.
- **Mayor flexibilidad**: el bajo acoplamiento proporcionado por los frameworks aporta una gran flexibilidad, que responde a los cambios de los requisitos tanto del negocio como de tecnología.
- **Arquitectos más eficaces**: el esqueleto o estructura proporcionado por los frameworks permite que los arquitectos dediquen más tiempo a la toma de decisiones imprescindibles en vez de gastar una enorme cantidad de tiempo en hacer cumplir las mejores prácticas (el framework hace el trabajo).

- **Líderes más eficientes**: el mayor desafío para un líder de proyecto es organizar el trabajo correctamente y asignar los recursos adecuados a las personas idóneas, decisiones que se facilitan extraordinariamente dentro de la estructura proporcionada por el framework.

Producción y mantenimiento:

Los frameworks aportan ventajas a la producción y al mantenimiento debido a la flexibilidad y consistencia que brinda su empleo correcto. Si usamos un framework de forma adecuada el software puede responder fácilmente a los cambios en los requerimientos tanto del negocio como de tecnología. El mantenimiento de las aplicaciones históricamente ha requerido descubrir cómo un desarrollador solucionó un problema determinado y luego cambiar ese comportamiento. Una solución a esto es usar miembros del equipo de desarrollo original para mantener el software, que evidentemente no hace un uso eficiente de los recursos. En cambio al usar un framework, una solución consistente es usada a través del desarrollo y por ende lleva consigo uno de los principios fundamentales de los frameworks, la flexibilidad. Es muy fácil localizar la parte de la aplicación afectada por los nuevos requerimientos y realizar los cambios de una manera controlada.

Desafíos.

Cuando son usados junto a los patrones de diseño, las librerías de clases y los componentes, los frameworks pueden incrementar significativamente la calidad del software y reducir el esfuerzo de desarrollo. Sin embargo, existen innumerables retos que deben ser aceptados con el objetivo de emplear eficientemente un framework. Cuando se arriba a la decisión de construir o usar a gran escala un framework con frecuencia se fracasa.

Existen un número de desafíos en la aplicación exitosa de un framework; entre los que se destacan: curva de aprendizaje, esfuerzo de desarrollo, mantenimiento, integración, validación y eliminación de errores y la falta de estándares.

- **Curva de aprendizaje**: la adquisición del conocimiento para usar eficientemente un framework orientado a objetos requiere un esfuerzo considerable por parte de los miembros del equipo de desarrollo. Con frecuencia se requiere de 6 a 12 meses para crear desarrolladores altamente productivos, en dependencia de la experiencia de los involucrados. Usualmente se necesita de cursos de entrenamiento y tutores para que enseñen a los desarrolladores para explotar al máximo las potencialidades del framework.
- **Esfuerzo de desarrollo**: el desarrollo de aplicaciones complejas resulta suficientemente difícil y desarrollar frameworks con alta calidad, extensibles y reusables es aún más difícil. Las habilidades que requiere el desarrollo satisfactorio de un framework con frecuencia permanecen entre los más expertos desarrolladores.
- **Mantenimiento**: los requerimientos de una aplicación cambian con mucha frecuencia; por consiguiente, los requerimientos de un framework también cambian a menudo. Como los frameworks evolucionan regularmente, las aplicaciones construidas sobre ellos también pueden evolucionar con facilidad.
- **Integración**: la rapidez y eficiencia en el desarrollo de aplicaciones crecerá en proporción directa con la integración de múltiples frameworks, por ejemplo, frameworks para el manejo de la interfaz gráfica de usuario, sistemas de comunicación, bases de datos, etc. así como el uso de librerías de clases, experiencias del desarrollo de otros sistemas y el re-uso de componentes existentes. Sin embargo los frameworks, en ocasiones, presentan problemas de integración. Estos problemas pueden aparecer en varios niveles de abstracción, desde la creación de la documentación (**Fayad, 97**), hasta la arquitectura de despliegue y distribución.
- **Validación y eliminación de errores**: a pesar de que un framework modular y bien diseñado puede localizar el impacto de los defectos de una aplicación, la validación, la detección y corrección de errores puede ser una tarea muy compleja por las siguientes razones:

1. Los componentes son muy difíciles de validar de forma abstracta.
2. La inversión de control y la ausencia de un control explícito del flujo de eventos.

- **Falta de estándares**: actualmente no existe un estándar ampliamente aceptado para el diseño, implementación, documentación y adaptación de los frameworks.

1.5 Proceso de desarrollo de los frameworks

Los frameworks son la piedra angular de la moderna ingeniería del software. El desarrollo de frameworks está ganando rápidamente una amplia aceptación debido a su capacidad para promover la reutilización del diseño y del código fuente (**Markiewicz, 01**). El proceso de desarrollo de un framework es diferente al aplicado para la creación de una aplicación estándar. Probablemente la distinción más importante es que el framework debe cubrir todos los conceptos relevantes dentro de un dominio, mientras que una aplicación sólo debe preocuparse por aquellos conceptos relacionados a los requerimientos del negocio que está modelando. Existen varias propuestas documentadas para desarrollar un framework. La mayoría de ellas, sin embargo, coinciden en que el objetivo es identificar las abstracciones con un acercamiento desde lo sencillo a lo complejo, es decir, comenzar examinando las soluciones existentes y luego generalizarlas.

1.5.1 Propuesta de Taligent

Esta propuesta se enfoca en tratar al framework como un producto de software (**Taligent, 95**) y recomienda:

- Derivar el framework de los problemas y soluciones existentes.
- Desarrollar el framework dividiéndolo en partes más pequeñas.
- Construir el framework usando un proceso iterativo conducido por la participación de los usuarios finales y creando prototipos.

Según (**Taligent, 94**) este proceso de desarrollo se divide en cuatro fases: identificar y caracterizar el dominio - análisis del dominio en (**Markiewicz, 01**) - definir la arquitectura y el diseño, implementación y despliegue. En cada una de estas fases se realizan las siguientes actividades:

Identificar y caracterizar el dominio:

1. Esbozar el proceso.
2. Examinar las soluciones existentes.
3. Identificar las principales abstracciones.
4. Identificar de qué parte del proceso el framework sería responsable.

Definir la arquitectura y el diseño:

1. Centrarse en cómo los usuarios interactúan con el framework. Para ello se debe conocer:
    a. ¿Qué clases el cliente instancia?
    b. ¿De qué clases el cliente hereda?
    c. ¿Cuáles son las funciones y tareas que el cliente utiliza?
2. Aplicar patrones de diseño.
3. Intercambiar ideas con el cliente y refinar la propuesta.

Implementar el framework:

1. Implementar las clases bases.
2. Iterar para refinar el diseño e incorporar características y funcionalidades al framework.

Despliegue:

1. Proporcionar documentación del framework.
2. Facilitar ayuda técnica a los usuarios.
3. Mantener y actualizar el framework.

En la propuesta hecha en (**Taligent, 95**) se introducen algunos conceptos interesantes en la fase de diseño, como es el caso de los patrones de diseño, discutidos con anterioridad en este capítulo, y que dan solidez al framework; otro idea es la de proporcionar documentación (ejemplos) y ayuda técnica a los usuarios.

1.5.2 Catalysis

En (**D´ Souza, 99**), se define el método Catalysis, donde se elabora una caracterización de su naturaleza paralela e iterativa, la atención continua al aseguramiento de la calidad y las pruebas y refiere la necesidad del énfasis temprano en la arquitectura.

A grandes rasgos, se puede definir el proceso de la siguiente manera:

- **Modelo del negocio** (**dominio**): este modelo describe los procesos que ocurren en el negocio en cuestión. Estos procesos pueden ser interacciones entre personas o empresas, procesos físicos, o el diseño de un sistema informático existente. La idea es conseguir una comprensión clara y precisa de los conceptos y reglas importantes para todos los interesados, especialmente los usuarios y los desarrolladores.
- **Contexto del sistema y especificación de requerimientos:** exige una clara comprensión del papel que el sistema que deseamos construir jugará en el contexto del negocio; las interacciones que tendrá con otros sistemas, las funcionalidades que debe poseer – especificadas en los requisitos funcionales - , y las prestaciones, tales como, seguridad, apariencia, etc. – especificadas mediante los requisitos no funcionales.
- **Diseño y especificación de componentes**: en esta fase se diseñan los componentes, así como sus dependencias; se elabora un modelo del contexto a través de casos del uso: entender las colaboraciones entre los objetos y se deben utilizar diagramas de estado.
- **Implementación, prueba y despliegue**: las actividades para construir, probar e instalar un sistema están pensadas en varios niveles: puesta en práctica del diseño, de las especificaciones del sistema y del modelo del negocio (dominio).

Catalysis recomienda usar un framework para construir otro.

1.5.3 Propuesta de Riehle

La propuesta realizada en (**Riehle, 00**) es una extensión evolutiva del modelado de frameworks basado en clases; aunque añade los conceptos de modelado basado en roles. En este método, basado en roles, los objetos juegan roles que son descritos por los tipos de roles y usualmente un objeto desempeña varios de ellos, por lo que la clase del objeto está compuesta por varios tipos de roles.

El modelado por roles para el diseño de frameworks define éstos, como artefactos de diseño e implementación explícitos con límites bien definidos, es decir, enmarcados en un dominio específico.

En (**Riehle, 00**) se precisan los siguientes conceptos:

- Rol: aspecto observable del comportamiento de un objeto.
    - Un objeto que proporciona un rol particular, se dice que desempeña dicho rol.
    - Un objeto puede desempeñar varios roles.
- Tipo de rol: define el comportamiento de un rol que un objeto puede desempeñar. Define las operaciones y el modelo de estado del rol, así como la semántica asociada al mismo.
    - El comportamiento de un objeto está definido por la composición de todos los tipos de roles de todos los roles que puede desempeñar.
- Tarea de colaboración: representa una actividad de propósito simple en una colaboración de objetos; que realizan desempeñando los roles definidos por la tarea o actividad.
- Modelo de roles: es un conjunto de tipos de roles que se relacionan unos con otros por medio de contratos o reglas, así como las relaciones que se establecen entre objetos.

Además revisa y extiende las bases del modelado de objetos, para soportar de forma eficiente los conceptos del modelado por roles. Presenta una nueva perspectiva en el concepto de rol, calificándolo como una necesidad para el diseño y documentación de los frameworks. Aplica los conceptos del modelado basado en roles al diseño y desarrollo de frameworks; plantea la extensión del framework a través de la herencia, la división por capas para ayudar al diseño de las aplicaciones, y la documentación del framework.

Esta propuesta para desarrollar frameworks combina los puntos fuertes del modelado por roles con los del modelado basado en clases, mientras deja a un lado las debilidades de este último. Es, por tanto, una extensión evolutiva de las propuestas actuales, y preserva las inversiones existentes.

1.5.4 Propuesta de Markiewicz y Lucena

Según (**Markiewicz, 01**) las cuatro etapas fundamentales del desarrollo de un framework son análisis del dominio, diseño, implementación e instanciación, como muestra la **Figura III**.

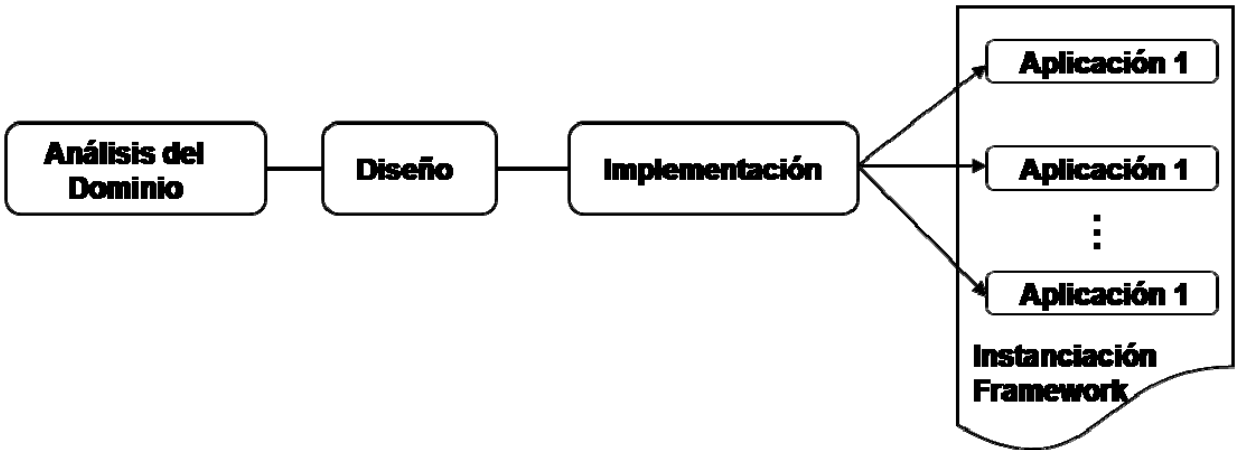

Figura III Proceso de desarrollo de los frameworks.

Análisis del dominio.

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.

A diferencia de la fase de captura de requisitos de los sistemas tradicionales, 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, se 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 resultarán excesivos; es más, será necesario tener individuos familiarizados con el dominio, con los prototipos y/o con los sistemas de software similares. No obstante, encontrar la experiencia que cubra un dominio grande es bastante difícil.

Por otra parte, si se 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, un framework llega a ser más maduro, cambia y se desarrolla. Este proceso puede representar la alteración de su arquitectura de configuración, pues aparecen nuevos requerimientos que no tenían soporte. Mientras tanto, las aplicaciones generadas usando el framework también evolucionan y cambian.

Técnicas para el análisis del dominio, tales como FODA (**Kang, 90**), **F**utured-**O**riented **D**omain **A**nalysis, se centran en encontrar los requerimientos para un dominio completo más que para una aplicación específica. El proceso de análisis del dominio, según (**Arango, 91**), identifica los conceptos y las relaciones que se establecen entre dichos conceptos dentro de un dominio específico. El análisis ayuda a identificar las partes que pueden variar, y aquellas que permanecen inalterables dentro del dominio, así como las abstracciones primarias necesarias para el desarrollo del framework. Según **FODA**, el análisis del dominio consiste en extraer de diferentes aplicaciones que

pertenezcan al dominio las partes que las hacen diferentes, es decir, hacer una abstracción de las partes comunes a dichas aplicaciones. Los conceptos encontrados serían entonces las partes invariables del framework.

Diseño

El diseño de un framework es un proceso iterativo que requiere tanto conocimiento del dominio como experiencia en el proceso de ingeniería del software. Debe comenzar con un esqueleto o una arquitectura base que capture la visión del framework basado en la comprensión de la jerarquía de las abstracciones que describen el dominio. Iterativamente enriquecer esta base agregando componentes adicionales que encapsulen los conceptos y funcionalidades relacionados con el dominio encontrados en la fase anterior. Se modelan los puntos calientes y los puntos congelados, quizás con un diagrama UML, así como la extensión y la flexibilidad propuesta en el análisis del dominio

Un framework debe ser lo suficientemente simple como para que pueda ser aprendido fácil y rápidamente por los desarrolladores que lo usarán. Debe incorporar una teoría del dominio así como una solución para un conjunto de problemas dentro de dicho dominio.

Implementación

En la fase de implementación se codifican los puntos congelados y los puntos calientes que pueden ser instanciados por las aplicaciones que utilizarán el framework, así como los que deben ser extendidos a partir de las clases y métodos abstractos definidos en el framework.

Instanciación

Finalmente, en la fase de instanciación, los puntos calientes del framework son implementados, generando una aplicación concreta dentro del dominio. Es importante observar que cada una de estas aplicaciones tendrá los puntos congelados del framework en común.

Los frameworks se deben ver como un producto de software (**Taligent, 94**), por lo que se debe aplicar un proceso similar para su desarrollo. En la propuesta hecha en (**Markiewicz, 01**), surge la necesidad de aplicar un proceso para caracterizar, evaluar y documentar el framework desarrollado.

1.6 Conclusiones parciales

En este capítulo se ha hecho una fundamentación teórica de los frameworks y se ofrece una clasificación según su extensibilidad que resumimos a continuación:

- Frameworks de **caja blanca**: la instanciación solamente es posible a través de la creación de nuevas clases que instancien el framework por herencia o composición, lo que implica que el usuario debe conocer a plenitud la arquitectura interna del framework.
- Frameworks de **caja negra**: producen instancias usando escrituras o scripts de configuración. Después de la configuración, una herramienta automática de instanciación crea las clases y el código fuente correspondiente.

Y según el dominio de problemas a que van dirigidos, fueron clasificados en:

- **Frameworks de aplicación**: encapsulan una capa de funcionalidad horizontal que se puede aplicar en la construcción de una gran variedad de aplicaciones.
- **Frameworks de soporte**: proporcionan servicios básicos a nivel de sistema.
- **Frameworks de dominio**: aplicables a un dominio de aplicación o a una línea de productos, como aplicaciones bancarias, tráfico, etc.

Se presentaron los beneficios y desafíos que representa usar un framework. Se han caracterizado otras propuestas para la reutilización tanto del diseño como de las implementaciones, tales como los patrones de diseño, los componentes, las librerías de clases, y una promesa muy reciente, los contenedores. Y por último se revisaron algunas de las propuestas para el desarrollo de los frameworks, específicamente las presentadas en (**Taligent, 95**), (**D' Souza, 99**), (**Riehle, 00**) y (**Markiewicz, 01**). Aunque divergen en algunos puntos hacen referencia a un conjunto de tareas o fases

que debe cubrir este proceso, como análisis del dominio, diseño y definición de una arquitectura base, partiendo de soluciones ya presentadas a problemas dentro del dominio en cuestión, implementación y despliegue del framework. En (**Riehle, 00**) se definen los conceptos relacionados con el diseño de frameworks basado en el modelado por roles; se presentan algunas extensiones a las tecnologías actuales para soportar tales conceptos como extensiones a lenguajes de programación y al UML.

Relacionado con el proceso de desarrollo de los frameworks es interesante observar que, como se precisa en (**Opdyke, 90; Opdyke, 92**), una de las características principales de un framework es que está diseñado para ser refinado, los buenos frameworks son generalmente el resultado de varias iteraciones de diseño y mucho trabajo, que en ocasiones implica realizar cambios estructurales.

CAPÍTULO 2

PROCEDIMIENTO PROPUESTO

2.1 Introducción

Los frameworks proporcionan una gran capacidad para reutilizar los artefactos de diseño e implementación. El punto de mira de la mayoría de los desarrolladores de los frameworks, así como de los usuarios de éstos, han sido las pruebas de las implementaciones contenidas en ellos. Los desarrolladores de sistemas de alta seguridad (aunque deberían ser todos), requieren más que simplemente una implementación; requieren una elevada confianza en la exactitud de cualquier componente de software que usen. Las técnicas de validación tradicionales, como las pruebas unitarias, etc., no proporcionan tal confianza.

En este capítulo se exponen los principales principios y/o propiedades que caracterizan a los frameworks. Además se presenta el procedimiento propuesto como solución a la problemática planteada en el diseño de la investigación.

2.2 Principios y/o propiedades

En los siguientes apartados se expondrán algunos de los principios y/o propiedades relacionadas con los frameworks (**Fayad, 97; Fayad, 00; Markiewicz, 01; Johnson, 04**). Se emplean indistintamente los términos principios y propiedades, entiéndase por ellos conceptos que caracterizan a los frameworks. Teniendo en mente desarrollar un framework bien diseñado, fácil de mantener y evolucionar, un elevado número de propiedades y características captarán nuestro interés. Por esta razón, en el presente trabajo se exponen las que consideramos esenciales, dejando para investigaciones futuras cubrir el amplio espectro de propiedades.

2.2.1 Alineación y pertenencia al dominio

Uno de los principios más importantes de un framework es que éste debe estar alineado con el conjunto de habilidades de los desarrolladores y los requerimientos del dominio que se está modelando. Un framework debe permitir a un equipo de

desarrolladores, no expertos en él, llevar de manera exitosa sus requisitos del negocio a las funcionalidades que satisfacen esos requerimientos. Esto sucede cuando el framework hace muy fácil y claro, en algunas ocasiones guiando al usuario paso a paso (frameworks de caja negra), a través del proceso de instanciación, dónde son implementados los requisitos específicos del sistema en desarrollo. Los frameworks son diseñados por expertos en el dominio, pero pueden ser utilizados por usuarios que no lo sean.

Esta propiedad expresa, además, que el framework debe asegurar que todas las aplicaciones construidas sobre él tienen que pertenecer al dominio. Para ello plantea que, si el dominio es demasiado grande, se desperdicia tiempo en recolectar y evaluar la información y sus recursos, además de que cubrirlo es bastante difícil. Por otra parte, si se elige un dominio demasiado estrecho, se reduce la su aplicabilidad y las aplicaciones generadas serán demasiado similares para justificar el esfuerzo de construirlo.

Durante el desarrollo del framework se debe tener en cuenta qué puntos calientes son obligatorios, cuales son los que realmente se necesitan en los requerimientos y los que son innecesarios o superfluos. Para garantizar con ello que todas las aplicaciones que instancien al framework, estén siempre dentro de los límites definidos en su dominio.

2.2.2 Patrones de diseño

Es importante mencionar qué son los patrones de diseño y cómo se aplican en la construcción de un framework. Los frameworks son un recurso valioso de las mejores prácticas de la industria; usando patrones de diseño se puede mejorar la calidad y consistencia del framework.

Según "*The American Heritage Dictionary*" (**American, 04**), un patrón es "un plan, diagrama, o modelo que se seguirá en la fabricación de cosas". Christopher Alexander dijo, "cada patrón describe un problema que ocurre repetidamente una y otra vez en nuestro ambiente, y después describe la base de la solución a ese problema, de una manera tal que Usted pueda utilizar esta solución en un millón de ocasiones, sin

emplearla de la misma manera dos veces" (**Alexander, 77**). Aún cuando Alexander hablaba acerca de patrones en la construcción de ciudades, lo que dijo también es aplicable en nuestro contexto. Nuestras soluciones están expresadas en términos de objetos e interfaces en vez de paredes y puertas, pero en la base de ambos tipos de patrones está una solución a un problema en un contexto.

Según (**Gamma, 95**), un patrón consta, por lo general, de cuatro elementos:

1. El **nombre del patrón** es un indicador que podemos utilizar para describir un problema del diseño, sus soluciones y consecuencias de forma breve. El nombramiento de los patrones nos permite diseñar en un nivel de abstracción mayor. Tener un vocabulario para los patrones permite hablar de ellos con los colegas, en la documentación, etc. Hace más fácil pensar en el diseño.
2. El **problema** describe cuando aplicar el patrón. Explica el problema y su contexto. Puede ser que describa problemas específicos del diseño tales como representar algoritmos como objetos, etc. Puede describir la estructura de la clase o del objeto que son sintomáticas de un diseño inflexible. En algunas ocasiones el problema incluye una lista de condiciones que deben ser resueltas para que tenga sentido aplicar el patrón.
3. La **solución** describe los elementos que mejoran el diseño, sus relaciones, responsabilidades y colaboraciones. La solución no describe un diseño específico o una implementación particular, porque un patrón es como una plantilla que puede ser aplicada en diversas situaciones. Un patrón proporciona una descripción abstracta de un problema de diseño y cómo una conjugación general de elementos (en nuestro contexto, las clases y los objetos) lo soluciona.
4. Las **consecuencias** son los resultados y las compensaciones de aplicar el patrón. Aunque las consecuencias, generalmente, no son expresadas cuando describimos decisiones del diseño, son críticas para la evaluación de alternativas, y para entender el costo y los beneficios de aplicar el patrón. Puesto que la reutilización es comúnmente un factor a tener en cuenta en el diseño

orientado a objetos, las consecuencias de aplicar un patrón incluye su impacto en la flexibilidad, extensibilidad o la portabilidad de un sistema.

Los patrones de diseño identifican los aspectos claves de una estructura común de diseño, que sirva de base para crear diseños reutilizables. Los patrones de diseño identifican las clases y las instancias que participan, sus roles y colaboraciones, así como sus responsabilidades.

Ahora, ¿cómo se aplican los patrones de diseño en el desarrollo de los frameworks? En el contexto de un framework, los patrones tienen diversos propósitos. Los frameworks deben aplicar patrones de diseño en su construcción así como en los problemas que soluciona. Algunos autores, plantean que un framework puede ser visto como la implementación de un conjunto de patrones **(Johnson, 97)**.

Ilustremos lo anterior con un ejemplo rápido. Supongamos que estamos usando más de un framework, digamos que uno para la navegación y uno para la persistencia, ambos podrían emplear los mismos patrones de diseño (por ejemplo: Facade, Flyweight y Singleton (**Gamma, 95**)) durante su construcción. Sin embargo debido a que tratan diferentes áreas de una aplicación, utilizan diferentes patrones. El framework para la navegación pudo utilizar el patrón Modelo Vista Controlador (MVC) (**Bushmann, 96**) mientras que el de persistencia pudo aplicar los patrones Representing Objects as Tables y Object Identifier (**Brown, 96**).

La aplicación de patrones de diseño en la construcción de un framework proporciona varios beneficios, incluyendo los siguientes:

- La puesta en práctica constantemente de los patrones de diseño será empleada por cada miembro del equipo de desarrollo, como oposición a cada persona usando el suyo. Por tanto las soluciones implementadas por varios desarrolladores tendrán más puntos en común que divergencias, lo que proporciona una mayor facilidad para el mantenimiento y la evolución del framework.

- Todos los miembros del equipo de desarrollo pueden resultar beneficiados del empleo de los patrones de diseños sin tener que ser expertos en éstos o gastar tiempo en llevar los conceptos a la práctica.

2.2.3 Inversión de control

La inversión de control es un fenómeno común que observamos al extender o desarrollar sobre un framework. De hecho, es considerado uno de los principios que lo define.

Consideremos el siguiente ejemplo. Imaginémonos que estamos escribiendo una aplicación para la inscripción de personas a un evento científico, y lo hacemos a través de una aplicación de consola. Sería algo como esto:

```
using System;

namespace InversionOfControl {
    public class IamTheBoss {
        static void Main(string[] args) {
            Console.Write("Entre su nombre: ");
            string nombre = Console.ReadLine();
            ProcesarNombre(nombre);
            Console.Write("Entre su e-mail: ");
            string email = Console.ReadLine();
            ProcesarEmail(email);
        }
        private static void ProcesarNombre(string name) {
            //Algún código aquí
        }
        private static void ProcesarEmail(string email) {
            //Algún código aquí, por ejemplo para validar que el
            //e-mail escrito sea correcto
        }
    }
}
```

Figura IV Aplicación de consola.

En este ejemplo, el código del desarrollador es el dueño del flujo de ejecución de la aplicación; decide cuándo hacer las preguntas, cuándo leer las respuestas y cuándo procesarlas.

Sin embargo, si fuera a desarrollar una aplicación de escritorio, tendríamos un código como el siguiente:

```
using System;
using System.Windows.Forms;

namespace FrameworkIsTheBoss {
    public class Formulario: Form {
        private Label lblNombre, lblEmail;
        private TextBox nombre, email;
        public Formulario() {
            lblNombre = new Label();
            lblNombre.Text = "Entre su nombre: ";
            this.Controls.Add(lblNombre);
            nombre = new TextBox();
            nombre.LostFocus += new EventHandler(nombre_LostFocus);
            lblEmail = new Label();
            lblEmail.Text = "Entre su e-mail: ";
            this.Controls.Add(lblEmail);
            email = new TextBox();
            email.LostFocus += new EventHandler(email_LostFocus);
        }
        private void nombre_LostFocus(object sender, EventArgs e) {
            ProcesarNombre(nombre.Text);
        }
        private void email_LostFocus(object sender, EventArgs e) {
            ProcesarEmail(email.Text);
        }
        private void ProcesarNombre(string name) {
            //Algún código aquí
        }
        private void ProcesarEmail(string email) {
            //Algún código aquí, por ejemplo para validar que el
            //e-mail escrito sea correcto
        }
        public static void Main(string[] args) {
            Application.Run(new Formulario());
        }
    }
}
```

Figura V Aplicación de escritorio.

Existe una diferencia considerable entre estos dos programas, precisamente el control de cuándo los métodos *ProcesarNombre(…)* y *ProcesarEmail(…)* son llamados. En la aplicación de consola, el desarrollador controla cuándo se hacen las llamadas a estos

métodos, en cambio, en la aplicación de escritorio no. En lugar de ello, el desarrollador entrega el control al framework mediante la llamada al método ***Application.Run(…)***. Es el framework el que decide cuándo llamar a los métodos implementados por el programador, basado en las asociaciones que se hicieron al crear la forma. Se invierte el control, "me llama algo que yo llamé", "el viejo código – el del framework, llama al nuevo código – el del desarrollador". A este fenómeno se le denomina "Inversión de Control", también conocido como el principio de Hollywood "no nos llame, nosotros le llamaremos".

Una característica importante de un framework es que los métodos definidos por el usuario para adaptarlo, frecuentemente son llamados desde el framework en sí, más que desde el código de la aplicación. El framework juega el rol de programa principal, coordinando y ordenando las actividades de la aplicación y controlando el flujo. Esta inversión de control le permite al framework servir como plantilla extensible. Las implementaciones provistas por el usuario adaptan los algoritmos genéricos definidos en el framework para una aplicación en particular.

La inversión de control es un elemento importante en qué hace diferente un framework de una libraría de clases. Una librería de clases es, esencialmente, un conjunto de funciones que el programador puede llamar, usualmente organizadas en clases. Cada llamada realiza una determinada tarea y retorna el control al cliente.

Un framework incorpora un determinado diseño abstracto. Para usarlo, el programador necesita insertar el nuevo comportamiento en varios lugares del framework, ya sea a través de la herencia o agregando nuevas clases. El código del framework entonces llamará al código del desarrollador en estos puntos, puntos calientes (hot spots) o de extensión, vistos en el **Capítulo 1**. Existen otras vías para lograr la inversión de control, como la mostrada en el ejemplo anterior, a través de la suscripción a eventos definidos por el framework; la plataforma .NET es un buen ejemplo de esto. Además, se puede hacer uso del patrón "Template Method", con el cual se define el esqueleto de un algoritmo para una operación, dejando para sus subclases la capacidad de redefinir el funcionamiento de los pasos de este algoritmo, siempre y cuando la estructura del

mismo permanezca intacta, de esta forma el framework llamará al código del desarrollador en los pasos del algoritmo que éste modificó, una vez ejecutado estos pasos, el control del programa retorna al framework.

2.2.3.1 Otro enfoque al principio Inversión de Control

Este enfoque del principio de inversión de control está expuesto en (**Johnson, 04**) y está estrechamente relacionado a los contenedores, descritos en el **Capítulo 1**. Sólo los objetos más simples encierran en sí toda la funcionalidad requerida; la mayoría de los objetos tienen dependencias. Por lo que necesitan de una forma para buscar y satisfacer estas dependencias aplicando buenas prácticas. Según (**Johnson, 04**) esto se puede lograr en la mayoría, sino en todos los casos, por medio de la inversión de control y la inyección de dependencias.

La inyección de dependencias es un tipo de inversión de control, que utiliza este principio para eliminar el código de búsqueda de dependencias, teniendo el contenedor que resolverlas automáticamente (**Johnson, 04**).

Estrategias de implementación

Según (**Johnson, 04**) la inversión de control puede ser implementada de dos formas diferentes:

- Operaciones de búsqueda de dependencias: el contenedor proporciona comunicación de vuelta a los componentes, y un contexto para las operaciones de búsqueda. Deja la responsabilidad a cada componente de usar las APIs del contenedor para buscar sus dependencias.
- Inyección de Dependencias: el contenedor es completamente responsable de resolver las dependencias, pasando estos objetos por medio de las propiedades o los constructores con parámetros. Al uso de las propiedades se le llama “Setter Injection”; y al uso de los constructores con parámetros se le denomina “Constructor Injection”.

La segunda de las estrategias, inyección de dependencias, usualmente es la mejor de las opciones. Hace al contenedor completamente responsable de la resolución de los

recursos, los objetos de negocio no tienen dependencia de las APIs del contenedor, ni deben realizar las operaciones de búsqueda. Por lo que el contenedor puede ser reconfigurado para usar un método diferente para obtener los recursos necesarios sin afectar el código de la aplicación.

La **Figura VI** (**Johnson, 04**) muestra las diferentes estrategias de implementación de la inversión de control.

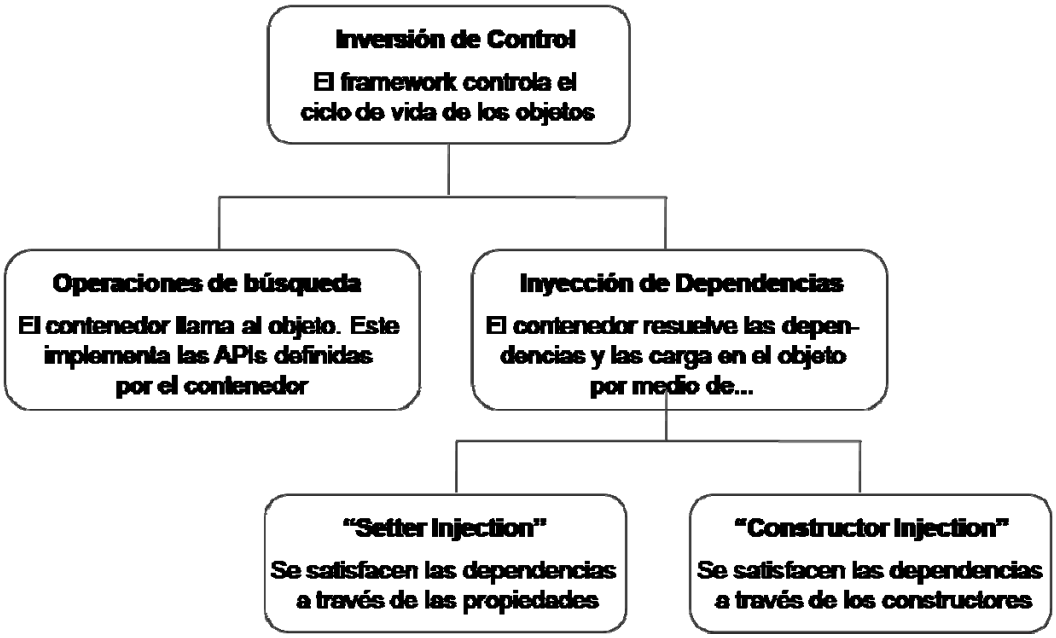

Figura VI Diferentes implementaciones del principio Inversión de Control.

Adoptando el principio de inversión de control, se hace más fácil incluir otros principios en los frameworks, por ejemplo, separación de intereses, bajo acoplamiento, extensibilidad, facilidad de configuración, etc.

2.2.4 Separación de intereses

Este principio se refiere a la necesidad de que el framework desarrollado posea una adecuada separación de las funcionalidades que implementa, que conste de una modularidad tal que facilite su inevitable mantenimiento y evolución.

Para ello, este principio hace referencia a la necesidad de resolver los siguientes problemas:

- **Código enredado** (**kickzales, 97**): una clase o módulo, además de implementar su funcionalidad principal debe ocuparse de otras competencias o intereses.
- **Código disperso** (**kickzales, 97**): el código que satisface una competencia está esparcido por distintas partes del sistema. Podemos distinguir dos tipos de código disperso:
    - Bloques de código duplicados: cuando los mismos bloques de código aparecen en distintas partes del sistema.
    - Bloques de código complementarios: cuando las distintas partes de una incumbencia son implementadas por módulos diferentes.

El **código disperso** y **enredado** es un código difícil de reutilizar, mantener y evolucionar y de pobre trazabilidad. Estos efectos hacen que disminuya la calidad del software diseñado y se reduzca la reusabilidad.

Las principales ventajas de aplicar este principio en la construcción del framework son:

- Se pueden asignar los desarrolladores de acuerdo con sus habilidades y experiencias, ganando en productividad.
- Los desarrolladores pueden trabajar simultáneamente en los diferentes componentes, y no de forma secuencial.
- Los componentes creados deben ser más reusables, ya que solucionan necesidades particulares, por ejemplo, seguridad, persistencia, entre otros; en oposición a la forma tradicional de que un mismo componente dé solución a varios intereses. Los componentes que tratan varios requisitos aumentan su complejidad, lo que los hace más difícil de entender y muy dependientes del escenario para el cual fueron creados. Estos efectos subyugan considerablemente la reusabilidad del código creado.

Una aplicación de este principio, que muestra claramente sus beneficios, es el patrón de diseño Modelo Vista Controlador (MVC) (**Bushmann, 96**). En pocas palabras, este patrón describe una solución para separar clara y eficientemente estas tres áreas relacionadas a la creación de interfaces de usuario. Esta separación permite que los desarrolladores trabajen simultáneamente dónde ellos son más productivos según sus habilidades, en la creación de las vistas, los modelos o los controladores. Las vistas y los modelos creados son muy reusables, y los tres pueden ser cambiados con un impacto mínimo en los demás.

Ejemplos de frameworks que demuestran las ventajas que proporciona el uso de este principio lo constituyen, Spring, que incluye su propio framework para trabajar con la Programación Orientada a Aspectos (AOP) (**Kickzales, 97**), que es una de las soluciones más elegante al problema de la separación de intereses y las incumbencias transversales; el caso de la seguridad en ASP.NET; etc. Además existen una gran variedad de implementaciones de los conceptos relacionados a la Programación Orientada a Aspectos, varios de ellos libres.

Comparando gráficamente el mismo framework antes y después de aplicar este principio:

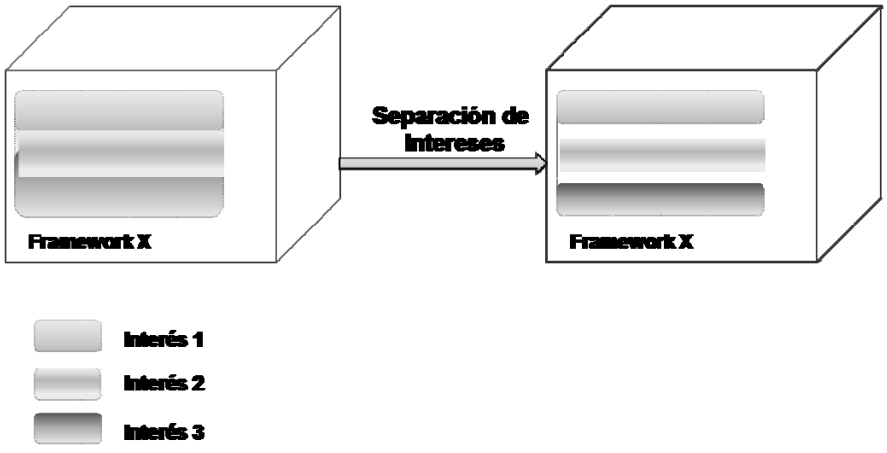

Figura VII Principio de separación de intereses.

2.2.5 Extensibilidad

La extensibilidad es la capacidad de cambiar o mejorar perceptiblemente el comportamiento del framework mediante la incorporación de nuevas clases para que éste use. Estas clases pueden heredar de otras existentes en el framework o implementar las interfaces definidas por el mismo. La **Figura VIII** muestra lo expuesto anteriormente:

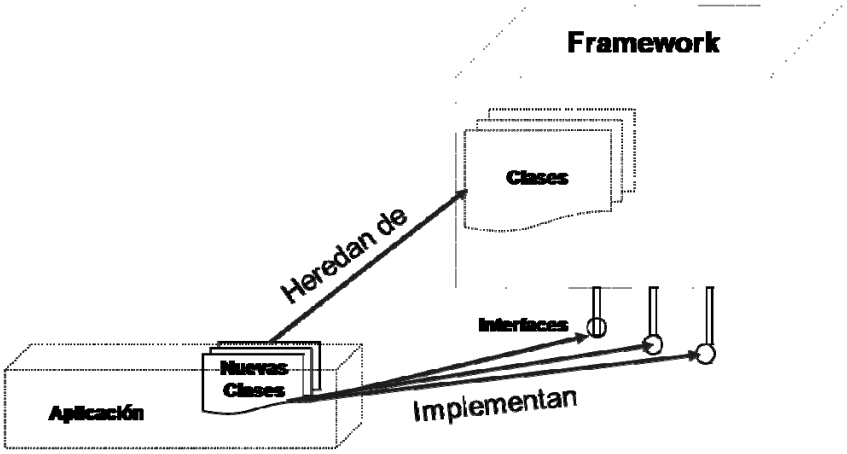

Figura VIII Principio de extensibilidad.

Por medio de la extensibilidad el framework permite hacer cambios en su comportamiento que puedan responder a los requisitos particulares de una aplicación ahora o en el futuro. Este principio es crítico si se desea que el framework tenga un largo período de vida. No importa cuán completo y exacto sea el framework en la satisfacción de los requisitos actuales del dominio, estos requisitos están destinados a cambiar.

2.3 Presentación del procedimiento propuesto

A continuación se describe el procedimiento propuesto. Haciendo énfasis en, su definición y esquema general y, se detallan, además, las fases del mismo.

2.3.1 Definición

El procedimiento propuesto constituye una guía para la evaluación y caracterización de las principales propiedades de los frameworks. Se fundamenta en el marco teórico descrito en el **Capítulo 1**, donde se argumenta la necesidad de un proceso para caracterizar y evaluar los frameworks; así como en el presente capítulo, donde se hizo una exposición no exhaustiva de los principios y/o propiedades que los caracterizan.

El procedimiento tiene como objetivo evaluar y caracterizar los frameworks; exponer a los desarrolladores las principales propiedades y/o principios que sustentan su construcción, evolución y documentación satisfactoria; así como revelar defectos u omisiones de alguna de estas propiedades en durante el desarrollo o evolución del framework.

La propuesta consta de un proceso lógico de fases a seguir, que abarcan la totalidad de las propiedades y/o principios expuestos con anterioridad, permitiendo, además, la inclusión de nuevos atributos y, de un conjunto de subactividades o actividades específicas. En particular aporta los siguientes elementos:

1) La definición de un **marco conceptual** que dará soporte a la evaluación del framework.
2) **Actividades generales** para cubrir el rango de principios o propiedades que caracterizan (no definen) a los frameworks.
3) **Actividades específicas** para la determinación, caracterización y valoración de estas propiedades.

2.3.2 Esquema General

En la **Figura IX** se muestra el esquema general del procedimiento, señalando las fases y actividades generales.

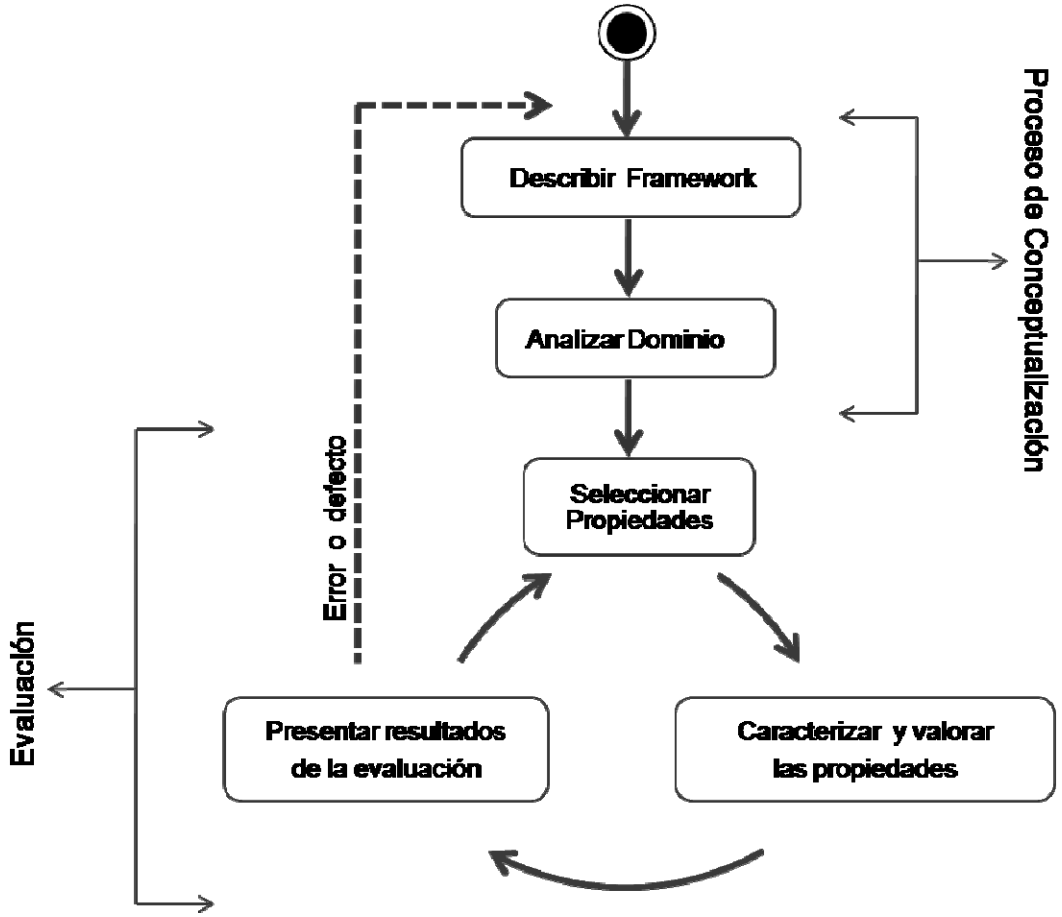

Figura IX Esquema general del procedimiento.

El procedimiento propone dos fases generales:

1) **Proceso de conceptualización**: tiene como objetivo definir el marco conceptual, describir y caracterizar el framework a evaluar. Constituye el punto de partida y la principal entrada para las actividades de evaluación.

2) **Evaluación**: en esta fase se seleccionan y priorizan las propiedades a evaluar en cada iteración. Se desarrollan las actividades necesarias para evaluar dichas propiedades. Se exponen los resultados del proceso y se elaboran las estrategias necesarias para mitigar los defectos o errores de concepción del framework (en caso de que exista alguno).

El procedimiento posee la característica de ser iterativo; lo que permite a los desarrolladores corregir errores de diseño, implementación o de conceptos que puedan aflorar durante su aplicación. El número de iteraciones de la fase de evaluación estará determinado por el número de propiedades a evaluar, teniendo en consideración que existen propiedades que pueden ser evaluadas simultáneamente y otras que deben ser priorizadas, ya que brindan soporte al proceso de evaluación.

2.3.3 Artefactos y miembros involucrados.

Para el desarrollo de un framework, se puede usar alguna de las propuestas expuestas en el **Capítulo 1** o combinar eficientemente la experiencia con alguna de las metodologías de desarrollo de software como RUP, XP, ICONIX, etc. Ahora, un punto común entre todas estas opciones lo constituye el uso de UML como lenguaje de modelado estándar, en algunas propuestas con extensiones específicas. Por lo que se propone emplear durante la aplicación del procedimiento los artefactos UML generados durante el desarrollo o evolución del framework.

Entre los artefactos UML cabe citar los siguientes:

- Diagrama de Casos de Uso del Negocio – Modelo de Objetos del Dominio.
- Diagramas de Actividades.
- Diagramas de Secuencia.
- Diagramas de Colaboración.
- Diagramas de Componentes.
- Diagramas de Despliegue.

En los cuales quedan recogidos los conceptos relacionados al framework modelado. Mostrando tanto aspectos estáticos como dinámicos del sistema. Existen otros artefactos que pueden ser de gran utilidad para aplicar el procedimiento propuesto como son las especificaciones de los requerimientos del dominio, los requerimientos funcionales y los no funcionales, además, las especificaciones de la arquitectura.

En la aplicación del procedimiento pueden estar involucrados tantos miembros del equipo de desarrollo como se considere necesario. La realidad nos sugiere que el procedimiento no necesita implicar a cada miembro en cada iteración y actividad. Dependiendo del tamaño, importancia, y complejidad del sistema, el número de involucrados en el proceso puede ser mayor o menor. Si el sistema tiene un número elevado de requisitos, o se está modelando un dominio con un alto grado de complejidad, se debe contar con un número considerable de expertos.

Debido a la complejidad e importancia del proceso de conceptualización para la correcta aplicación del procedimiento, sugerimos que en esta actividad estén involucrados la mayor parte de los miembros del equipo de desarrollo así como posibles usuarios finales. Durante las iteraciones del proceso de evaluación se requiere la atención del arquitecto principal y los desarrolladores con mayor experiencia, pues en esta fase se deben tomar las decisiones claves que garanticen la calidad y solidez del framework. Terminando con la atención de todos los involucrados en el desarrollo del framework para presentar los resultados del proceso de evaluación.

2.3.4 Primera fase: proceso de conceptualización

Para evaluar y caracterizar los frameworks resulta necesario establecer un marco conceptual de trabajo concreto, que permita analizar los aspectos más relevantes en relación al framework en desarrollo, y que darán soporte al proceso de evaluación. El punto de partida para definir este marco es el contexto teórico de referencia –abordado con profundidad en el capítulo anterior- del cual se han extraído, integrado y relacionado los principales conceptos. Este marco conceptual asegura las premisas sobre las cuales se realizarán las iteraciones del proceso de evaluación.

Para asegurar la efectividad y objetividad de esta fase del proceso se proponen las actividades específicas que se detallan a continuación:

1. Descripción del framework.
    a. Objetivos del framework.

b. Elementos y condiciones que sustentan y justifican la concepción y construcción del framework.

c. Metodología y herramientas de desarrollo empleadas.

2. Análisis del dominio en el cual se enmarca el framework.

   En esta actividad se debe definir el alcance del dominio, o sea sus límites; de manera que resulte evidente si un requisito está o no dentro de éste. Por otra parte resulta una herramienta importante para decidir si una aplicación debe o no utilizar el framework. En esta actividad se debe elaborar el modelo de objetos del dominio o el modelo de casos de uso del negocio.

Durante esta fase quedan registrados los conceptos más relevantes que apoyarán el desarrollo de las actividades durante las iteraciones de evaluación.

2.3.5 Segunda fase: evaluación

Durante esta etapa se deben seleccionar y priorizar cuidadosamente las propiedades a evaluar, determinando con cuales se procederá en cada una de las iteraciones. Existen varias propiedades o principios deseables en un framework, que si bien no resultan imprescindibles sí son muy convenientes, tales como: alineación y pertenencia al dominio, patrones de diseño, extensibilidad (**Fayad, 97; Fayad, 00; Markiewicz, 01; Johnson, 04**), etc., que fueron abordados en este capítulo. Estas son las principales propiedades a evaluar lo que no quiere decir que sean las únicas.

Teniendo en cuenta la importancia que tiene cada una de las propiedades en relación al éxito del framework, además de sus interdependencias, se propone la siguiente evolución del proceso iterativo (**Figura X**):

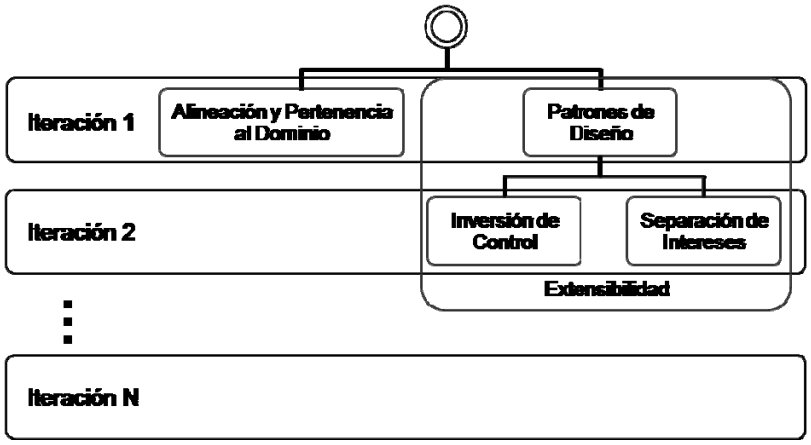

Figura X Evolución del proceso iterativo del procedimiento.

Como se aprecia en la **Figura X** existen propiedades que deben ser evaluadas en las primeras iteraciones, pues durante su evaluación emergerán detalles que contribuirán a la valoración de otras; tal es el caso de los patrones de diseño. De la misma manera algunas propiedades pueden ser evaluadas simultáneamente, considerando que tributarán unas a otras elementos complementarios para una evaluación más profunda; por ejemplo, el principio de "Inversión de Control" y el de "Separación de Intereses", que aunque no es imprescindible su presencia, inciden significativamente en la calidad del framework. En otros casos, la propiedad queda revelada casi completamente durante el proceso de evaluación de otras, así ocurre con la extensibilidad.

El procedimiento propuesto sugiere que en una primera iteración se evalúen dos propiedades de extraordinaria importancia para asegurar la solidez del framework desde el mismo comienzo; ellas son: alineación y pertenencia al dominio y patrones de diseños.

Para asegurar una adecuada evaluación se proponen las siguientes actividades:

- Valorar que al ser instanciado el framework, según la concepción elaborada durante el proceso de conceptualización, la aplicación desarrollada esté siempre dentro de los límites del dominio definidos en ese marco. Además, que exista

una correspondencia entre los requerimientos del negocio que el framework modela y las habilidades de los desarrolladores, pues el éxito del mismo no depende exclusivamente de su calidad, sino también de las posibilidades reales de explotación.

Una premisa de los framework consiste en facilitar y agilizar la conversión de los requisitos del negocio a las funcionalidades que satisfacen esos requisitos; si resulta más complejo aprender a usarlo que desarrollar desde cero, entonces no tiene sentido construirlo, a la decisión de su conveniencia o no es aplicable el análisis costo-beneficio.

Al contrario de los tradicionales ambientes de desarrollo de software orientado a objetos, un framework prescribe una estructura particular para las aplicaciones que lo instancian y proporciona los componentes y plantillas necesarias para construirlas. Los conceptos contenidos en el framework tienen un impacto directo en la estructura de las aplicaciones finales y en la forma en que éste es instanciado.

El framework permite a los desarrolladores de software construir aplicaciones completas, simplemente instanciando objetos estándares provistos por él y orientándolos conjuntamente para resolver un problema particular dentro del dominio.

Una ventaja del framework y que a su vez sustenta el rigor y alcance del mismo es que perfectamente se puede construir casi completamente una aplicación sin programar, por ejemplo, seleccionando íconos que representan los componentes y las estructuras estándares de las aplicaciones, conectándolas gráficamente y dejando que el sistema genere automáticamente una aplicación ejecutable (**Fugini, 92**).

- Caracterizar y evaluar los principales patrones de diseño que pueden ser empleados en la construcción del framework.

a. Definir detalladamente los patrones de diseño que pueden ser utilizados tanto en la construcción del framework como en las soluciones a los problemas del dominio que éste cubre.

b. Evaluar contextualmente los resultados y consecuencias de su uso.

La evaluación de los principales patrones de diseño constituye un punto de partida para la valoración de diversas propiedades de los frameworks, ya que en ocasiones éstas constituyen la implementación concreta de uno o varios patrones.

En una segunda iteración se procede a la evaluación de los principios “Inversión de Control” y “Separación de Intereses”, los que si bien no definen al framework, sí lo hacen más completo y reusable. Esta evaluación se puede realizar de forma simultánea.

- Evaluar la presencia o inclusión en el framework del principio “Inversión de Control”, lo que se realizará mediante las siguientes subactividades:

  a. Describir gráfica o textualmente el flujo de ejecución entre la aplicación y el framework.

  b. Definir la o las estrategias a seguir para la implementación de este principio.

  c. Evaluar la correcta implementación de dichas estrategias.

- Evaluar la presencia o inclusión en el framework del principio “Separación de Intereses”, lo que asegura que el framework no contenga “código enredado”, “código disperso” o “código interdependiente”; elementos que limitan significativamente la capacidad de reutilizar, mantener y evolucionar el framework.

  Esta acción está relacionada en gran medida con la última actividad de la iteración anterior y con la primera de la presente, por lo que los artefactos generados en estas actividades resultan de gran utilidad.

  Con este propósito resulta de conveniente verificar el uso de alguna estrategia para la separación de intereses.

Existen varias formas de resolver el problema de las dependencias transversales; una de ellas es aplicando patrones de diseño, por ejemplo:

- El patrón **Decorator** (**Gamma, 95**) permite anexar comportamientos a un objeto dinámicamente, ya sea antes o después de la invocación de un método. Este es un patrón de gran utilidad pero tiene el inconveniente que se debe definir y codificar una nueva unidad (decorador) por cada clase objetivo, lo que limita significativamente su aplicación en sistemas complejos.
- El patrón **Observer** (**Gamma, 95**) permite que múltiples objetos reciban notificación de acontecimientos sobre un objeto "observado".
- El patrón **Chain of Responsibility** (**Gamma, 95**) permite a más de un objeto, en una cadena, recibir las peticiones antes de que uno específico le dé respuesta.

Algunas tecnologías y frameworks admiten hacer intercepciones, permitiendo la adhesión de comportamiento adicional a los objetos en tiempo de ejecución. Una de las soluciones más elegante y reciente es la Programación Orientada a Aspectos (**Kickzales, 97**), AOP por sus siglas en inglés. No es objetivo del presente trabajo hacer un tratado del tema, pero cabe señalar algunas de las ventajas que brinda, así como varias de las estrategias para su implementación y que pueden, o deben, ser objeto de especial atención a la hora de desarrollar un framework.

Las estrategias más utilizadas para la implementación de los conceptos relacionados a la Programación Orientada a Aspectos, ordenadas en función de de su importancia son (**Johnson, 04**):

- Proxy dinámico.
- Generación dinámica de código.
- Uso de un cargador de clase personalizado.
- Extensiones al lenguaje.

La Programación Orientada a Aspectos aporta las siguientes ventajas (**Quintero, 00**):

- Un código menos enmarañado, más natural y más reducido.
- Más facilidad para depurar y hacer modificaciones en el código.
- Se tiene un código más reusable, que se puede acoplar y desacoplar cuando sea necesario.

Resultados de la evaluación

Finalmente, como última actividad de cada iteración está la presentación de los resultados de la evaluación. La información recogida durante el proceso de evaluación necesita ser resumida y presentada a todos los involucrados en el desarrollo del framework. Normalmente esta presentación se hace de forma verbal, con la utilización de diapositivas, pero en ocasiones, además, puede estar acompañada de un reporte más detallado y por escrito.

Como resultado del proceso de evaluación tenemos tres elementos que abordar:

- El procedimiento se ha aplicado correctamente.
- Las propiedades no sólo están presentes, sino bien implementadas.
- Un reporte detallado de los resultados de la evaluación.

El procedimiento debe ser aplicado correctamente, respetando cada fase y cada una de las actividades correspondientes. Realizando un análisis completo del framework a evaluar – proceso de conceptualización – y teniendo en cuenta que las propiedades seleccionadas para su evaluación no sólo deben estar presentes en el framework sino que deben estar correctamente implementadas.

Basado en la información recogida durante la evaluación se debe realizar un informe escrito que detalle esta información junto con cualquier estrategia propuesta para la mitigación de errores u omisiones. Este documento pasará a manos de los involucrados en el desarrollo del framework, quienes deben aplicar las estrategias propuestas para atenuar las limitaciones encontradas.

Aún cuando la mejora considerable de las especificaciones de la arquitectura y la implementación son consecuencias positivas de la aplicación del procedimiento propuesto, su objetivo es mostrar las decisiones claves en el desarrollo de un framework, que proporcione una idea clara y precisa de las principales propiedades que lo caracterizan.

Resulta conveniente identificar las decisiones significativas que pongan en riesgo la calidad del framework así como omisiones que limiten el alcance del mismo. Estas decisiones sirven de bandera roja “tener cuidado al cambiar o no incluir esta propiedad…”. De esta forma el procedimiento revela a los involucrados en el desarrollo del framework dónde enfocarse para obtener los resultados esperados.

2.4 Criterio de expertos

El método Delphi

En método Delphi es considerado como uno de los métodos subjetivos de pronósticos más confiables, constituye un procedimiento para confeccionar un cuadro de la evolución estadística de las opiniones de expertos o usuarios en un tema tratado. El mismo permite rebasar el marco de las condiciones actuales más señaladas de un fenómeno y alcanzar una imagen integral y más amplia de su posible evolución, reflejando las valoraciones individuales de los expertos, las cuales podrán estar fundamentadas, tanto en un análisis estrictamente lógico como en su experiencia intuitiva. Para el caso de los usuarios sus criterios están fundamentados en el valor de uso que encuentran en una propuesta determinada.

Elaboración de las encuestas

La encuesta elaborada (**Anexo B**) tiene preguntas abiertas que le permiten al sujeto consultado hacer una valoración crítica del tema, lo que constituye un elemento importante para conocer posibles limitaciones o insuficiencias del procedimiento propuesto, en cuanto a grado de relevancia de las fases y actividades**,** eliminación e inclusión de propiedades y sugerencias de cambios en algunos de los elementos que se discuten.

Selección de los expertos

Primeramente debemos señalar que, entendemos por experto tanto al individuo en sí como a un grupo de personas u organizaciones capaces de ofrecer valoraciones conclusivas de un problema en cuestión (el procedimiento propuesto) y hacer, además, las recomendaciones que considere válidas para su enriquecimiento.

Para la selección de los expertos se han tenido en cuenta las siguientes características:

- Competencia.
- Creatividad.
- Disposición a participar en la encuesta.
- Capacidad de análisis y de pensamiento.
- Espíritu colectivista.
- Espíritu autocrítico.

Dentro de las características de selección la **competencia**, que define el nivel de calificación del experto en una determinada esfera del conocimiento, resulta imprescindible; para su determinación se empleó la metodología descrita en (**Landeta, 99**).

Según (**Landeta, 99**) el procedimiento para la selección de los expertos se enmarca en cuatro etapas fundamentales:

- Determinación de la cantidad de expertos.
- Confección del listado de posibles expertos y determinación de su consentimiento para su participación en el peritaje.
- Estudio de las características de los posibles expertos y encuesta para determinar su coeficiente de competencia.
- Determinación final de los expertos que serán utilizados para valorar la propuesta.

Para ello se confeccionó un listado de **29** especialistas y después de cumplir el procedimiento para la selección de los expertos, **26** de ellos fueron seleccionados para la realización de la consulta, de los cuales el 90% (23) son profesores y el 7% (2) son Máster en Ciencias.

Análisis de los resultados

El **Anexo C** muestra los resultados generales de las encuestas. Como consecuencia de aplicar el método Delphi se obtuvo el siguiente resultado:

Pregunta 1: grado de relevancia de las fases y actividades del procedimiento.

CONCLUSIONES GENERALES					
	MR	BR	R	PR	NR
Describir Framework	Sí	-	-	-	-
Analizar Dominio	Sí	-	-	-	-
Selección de Propiedades	Sí	-	-	-	-
Caracterización y valoración de propied	Sí	-	-	-	-
Presentar resultados de la evaluación	Sí	-	-	-	-

Figura XI Resultados para la pregunta 1.

Pregunta 2: grado de relevancia de las propiedades básicas incluidas en el procedimiento.

CONCLUSIONES GENERALES					
	MR	BR	R	PR	NR
Alineación y pertenencia al dominio	-	-	Sí	-	-
Patrones de diseño	Sí	-	-	-	-
Inversión de Control	Sí	-	-	-	-
Separación de Intereses	-	Sí	-	-	-
Extensibilidad	Sí	-	-	-	-

Figura XII Resultados para la pregunta 2.

Las principales recomendaciones fueron:

- Construcción de una herramienta que automatice el proceso propuesto en la medida de lo posible.

- Inclusión de propiedades como: facilidad de configuración, bajo acoplamiento, performance, etc.

2.5 Conclusiones parciales

El desarrollo de un framework es un proceso complejo, que demanda una gran experiencia en el dominio que se está modelando y muy buenas prácticas en el desarrollo de software. Así lo demuestra el marco teórico analizado en el **Capítulo 1**. El cual aporta una valiosa información para abordar en un primer paso el estudio y análisis de los frameworks, y en un segundo paso elaborar el procedimiento propuesto en el presente trabajo.

Existen un gran número de propiedades y principios que se deben tener en cuenta a la hora de construir un framework. En el presente capítulo se han expuesto algunos de ellos:

- Alineación y pertenencia al dominio.
- Patrones de Diseño.
- Inversión de Control.
- Separación de Intereses.
- Extensibilidad.

Que por cuestión de tiempo no fueron más, pues esta lista pudiera crecer con, facilidad de configuración, bajo acoplamiento, facilidad de evolución, rendimiento, integración, consistencia etc. además de otras propiedades que dependerán del dominio que se esté modelando. El espectro de propiedades puede ser titánico, por lo que en el trabajo se expusieron las que consideramos son de vital importancia para un framework.

Además se presentó el procedimiento propuesto para la caracterización y evaluación de estas propiedades. El cual cuenta con dos fases generales, definición del marco conceptual y, evaluación. Estas fases a su vez, debido a sus objetivos, se dividen en varias actividades o pasos a seguir para la determinación, caracterización y valoración de estas propiedades. El procedimiento tiene un carácter iterativo, permitiendo la

inclusión de nuevas propiedades y, facilitando la rectificación de los defectos u omisiones encontrados, por medio de estrategias expuestas durante alguna de las iteraciones de la fase de evaluación.

Con el objetivo de conocer las opiniones de expertos y usuarios acerca del procedimiento propuesto se hizo uso de la técnica “Criterio de Expertos” empleando el método Delphi. El resultado obtenido evidencia que la efectividad y calidad de un framework está condicionada por la severidad con que se evalúen las propiedades que los caracterizan y la profundidad con que el equipo de desarrollo aborde el dominio que se modela. Por lo que el procedimiento propuesto constituye una herramienta importante para el desarrollo y evolución de los frameworks.

CAPÍTULO 3

APLICACIÓN DEL PROCEDIMIENTO

3.1 Introducción

El diseño de software es un proceso difícil, y el diseño de software correctamente implementado y reusable, es una tarea aún más difícil. Debido a la característica de los frameworks de cubrir un dominio entero de problemas, se hace realmente complejo su desarrollo. En el **Capítulo 2** se expusieron, de forma no exhaustiva, algunas de las propiedades o principios que caracterizan al framework, elementos que deben tenerse en cuenta desde las primeras fases del ciclo de desarrollo. Se presentó, además, una propuesta de procedimiento para evaluar estas propiedades, que sirva de base durante el desarrollo del framework, con el objetivo de asegurar la correcta implementación del mismo.

Con el objetivo de propiciar una visión más completa y práctica del procedimiento propuesto, así como un punto de partida para su posterior validación (no constituye un objetivo de este trabajo), el presente capítulo muestra su aplicación, no a un ejemplo, sino a un framework real, *freeTribe*. Este framework se encuentra en explotación en la Universidad de Holguín “Oscar Lucero Moya”. Los resultados de la aplicación del procedimiento servirán de guía para la evolución y desarrollo de futuras versiones del framework.

En primer lugar, el capítulo introduce el framework a abordar, *freeTribe*. A continuación y siguiendo el procedimiento propuesto, se evalúa el framework desarrollando cada una de las actividades. Finalmente concluye con las consideraciones y recomendaciones a tener en cuenta para futuras versiones de *freeTribe*.

3.2 freeTribe: framework para el desarrollo de sistemas colaborativos distribuidos

En atención a las difíciles condiciones a que se enfrentan los desarrolladores de groupwares para producir software colaborativo buenos y eficientes, se llevó a cabo una investigación que concluyó con el desarrollo de un framework para el desarrollo de sistemas colaborativos distribuidos, *freeTribe* (**FR**amework for d**E**v**E**lopment of dis**TRIB**uted groupwar**E**). El gran avance logrado en las tecnologías de la Informática, así como el número creciente de contribuciones en la disciplina de CSCW, sostienen la concepción de *freeTribe*.

El punto de partida del proceso de Ingeniería del Software de *freeTribe* lo constituye el Modelo Cooperativo propuesto en la metodología AMENITIES (un acrónimo para **A ME**thodology for a**N**alysis and des**I**gn of coopera**TI**ve syst**E**m**S**) (**Garrido 03**), donde varios artefactos UML son integrados según la metodología de Ingeniería del Software ICONIX (**Rosenberg, 05**). El lenguaje usado para dicha implementación del framework es Java, eso permite independencia de la plataforma y un mejor despliegue para los sistemas operativos más conocidos como Windows, Linux, Solaris y Macintosh.

3.3 Primera fase: proceso de conceptualización

En esta primera fase se establecen los conceptos esenciales para el desarrollo del proceso de evaluación.

Descripción del framework

Según (**Rodríguez, 06**) el objetivo principal del framework es facilitar la elaboración de aplicaciones distribuidas en ambientes heterogéneos.

En el propio texto (**Rodríguez, 06**) se hace un estudio del contexto en que se ubica el desarrollo de *freeTribe*, los costos y beneficios de su construcción, además de una valoración de sostenibilidad. Como resultado se tiene que el framework favorecerá considerablemente a los recursos humanos proporcionar una mayor producción de sistemas colaborativos distribuidos. Entretanto, en el ámbito tecnológico se puede decir que cualquier usuario puede interactuar con el framework sin necesidad de contar con

un alto nivel de conocimientos en las diferentes perspectivas del CSCW, porque para eso se concibió, para facilitar la difícil situación a que se enfrentan los desarrolladores de aplicaciones groupware distribuidas, además de que se dispone de la tecnología necesaria para implantarlo.

El desarrollo del framework está soportado por la metodología ICONIX, así como, por la propuesta metodológica AMENETIES para el desarrollo de las aplicaciones que instancien a *freeTribe*. Además las herramientas de *freeTribe* permiten automatizar el proceso de elaboración de los sistemas colaborativos distribuidos, agilizando el nivel de desarrollo de las aplicaciones colaborativas. *freeTribe* consta de dos herramientas, un administrador del servidor colaborativo y un servidor **derby** para alojar las bases de datos en un servidor remoto en caso de que se quiera.

Dominio en el cual se enmarca el framework

En (**Rodríguez, 06**) se define CSCW como el dominio de problemas que soporta *freeTribe*. Se expresa el alcance del framework en el siguiente gráfico:

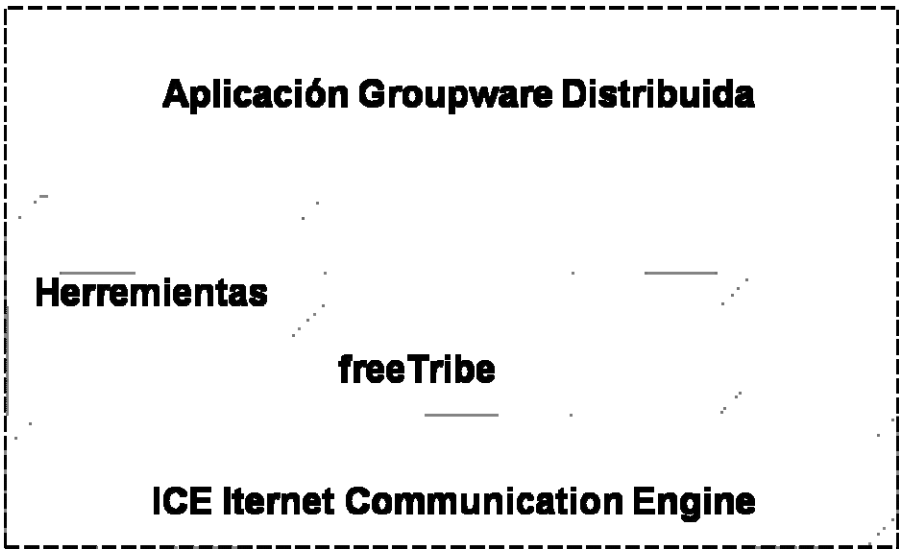

Figura XIII Visión de *freeTribe*.

Para estudiar y especificar los grupos de trabajo se debe establecer un marco conceptual de trabajo concreto que permita analizar los aspectos más relevantes del

sistema. Hacer una generalización de estos conceptos proporciona una visión general del dominio CSCW y a la vez se estará definiendo el modelo del domino de *freeTribe*.

Para realizar el modelo del dominio el autor (**Rodríguez, 06**) se basa en el marco conceptual de AMENETIES (**Garrido, 03**), extrayendo los siguientes conceptos significativos:

1. Acción: unidad básica de trabajo ejecutable automáticamente.
2. Subactividad: unidad de trabajo formada por un conjunto de acciones y otras subactividades que permite estructurar tareas.
3. Tarea: conjunto de subactividades/acciones cuya realización permite alcanzar objetivos.
4. Actor: usuario, programa o entidad que puede desempeñar roles.
5. Rol: comportamiento esperado de un actor en base a un conjunto identificable de tareas a realizar.
6. Capacidad: habilidad o responsabilidad asociada a un actor que le permite desempeñar roles y llevar a cabo tareas, subactividades o acciones.
7. Tarea Cooperativa: tarea en la cual para su realización interviene más de un actor, ya sea desempeñando el mismo rol o distintos.
8. Grupo: conjunto de actores desempeñando roles que pertenecen a una misma organización o que participan en la realización de tareas cooperativas.

Partiendo de los conceptos antes descritos se conformó el modelo del dominio de *freeTribe*, representado en la **Figura XIV**.

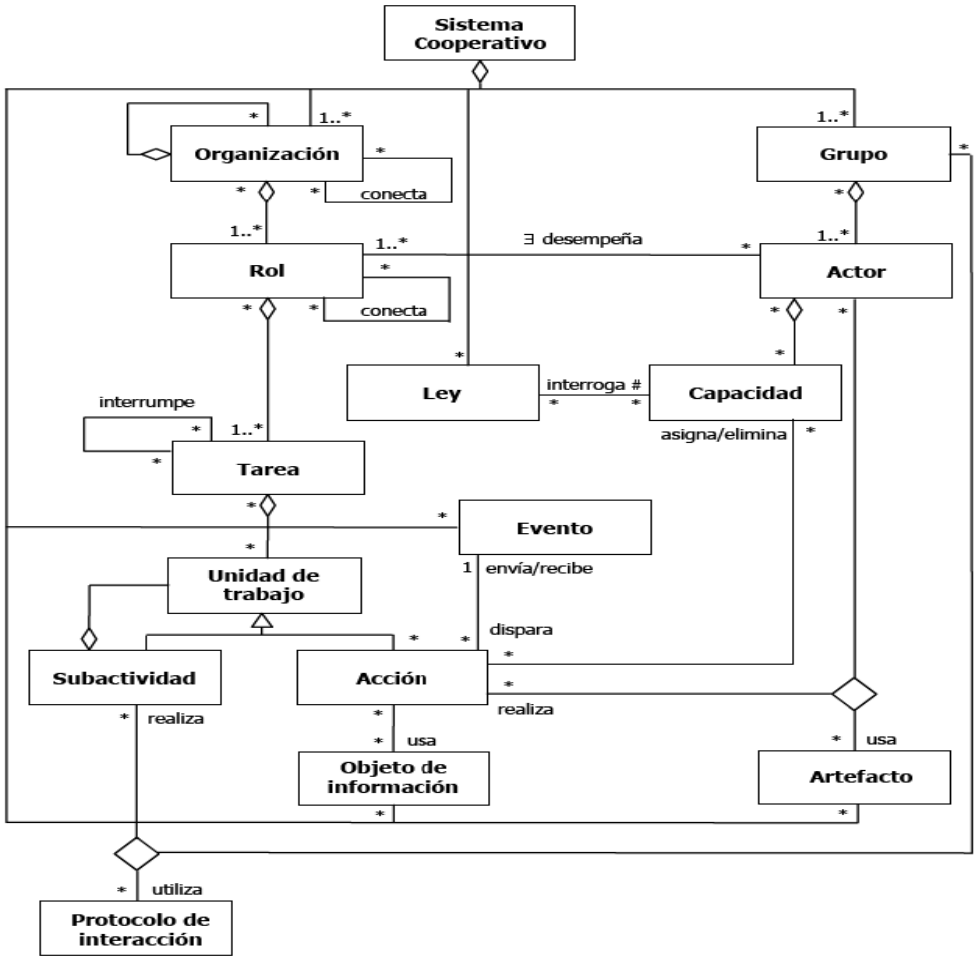

Figura XIV Modelo del Dominio de *freeTribe*.

Estructura del framework

La siguiente figura (**Rodríguez, 06**) muestra la arquitectura general de *freeTribe*.

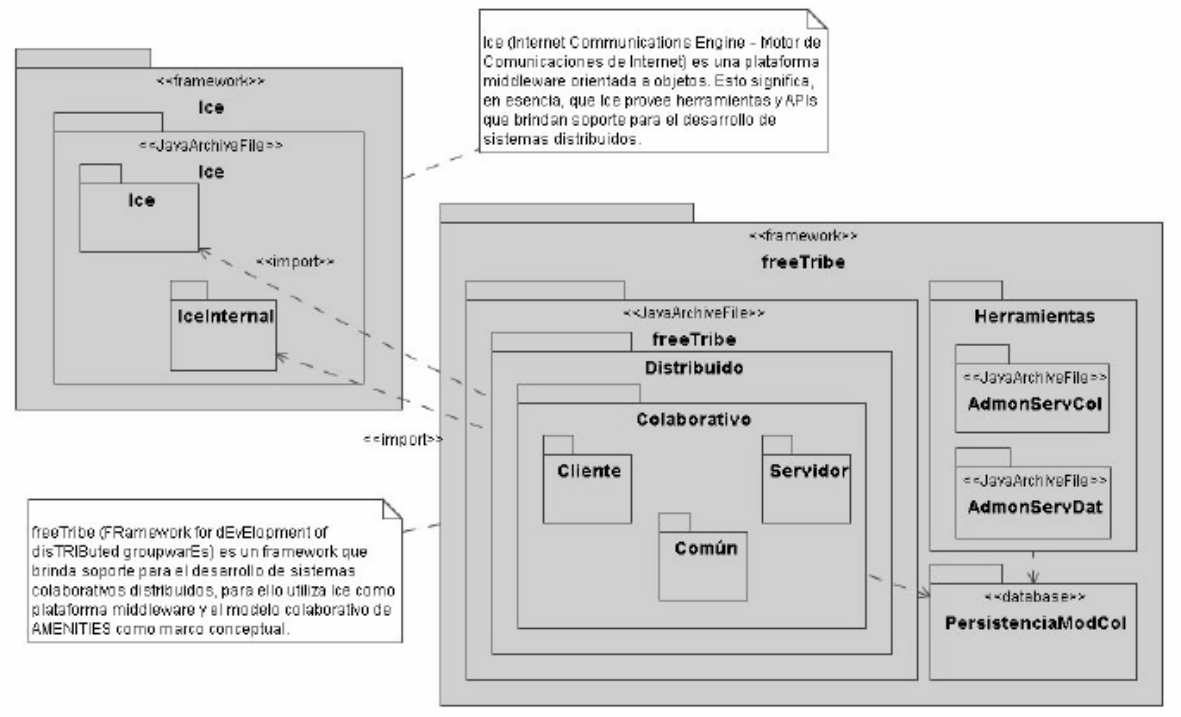

Figura XV Arquitectura general de *freeTribe*.

Todas las clases, de *freeTribe*, o sea, los puntos calientes y congelados, se encuentran encapsuladas en una biblioteca de clases que será necesaria para crear las aplicaciones groupware distribuidas que instancien dicho framework. Además, *freeTribe* consta de dos herramientas para agilizar dicho proceso de instanciación y de una base de datos interna que permite hacer persistente a los objetos correspondientes.

El framework consta de tres paquetes, que deben ser empleados indistintamente a la hora de instanciar y construir aplicaciones sobre *freeTribe*.

Paquete Servidor: este paquete contiene las clases que encapsulan la lógica del servidor colaborativo, mediante ellas las aplicaciones clientes pueden acceder de forma remota, a los servicios que brinda dicho servidor. Para ello el framework brinda los siguientes puntos de extensión:

- ClsActores: esta clase representa el concepto “actor” del modelo cooperativo de AMENETIES.

freeTribe representa todos y cada uno de los conceptos del modelo cooperativo de AMENETIES, elementos que constituyen, en su mayoría, puntos de extensión del framework.

- ClsPersistenciaObjetos: clase abstracta que permite definir la persistencia de los objetos y el almacenamiento en caché; así como otros elementos que permiten definir la forma en que se persistirán los objetos tanto en el servidor local como en uno remoto.

Paquete Cliente: este paquete contiene las clases que encapsulan la lógica de los clientes de sistemas colaborativos, además de las clases necesarias para acceder a las funcionalidades que brinda el servidor colaborativo.

- Clsdelegautentificacion: permite autenticar a un usuario, para que pueda tener acceso a las posibilidades que brinda la aplicación groupware.

freeTribe consta de una herramienta para dar soporte y agilizar la creación de aplicaciones distribuidas.

Paquete Común: contiene las funcionalidades comunes tanto al paquete **Servidor** como al **Cliente**.

Herramientas del framework

freeTribeBoss: permite automatizar el proceso de creación de un servidor colaborativo al brindar la posibilidad de definir los conceptos del modelo conceptual del modelo cooperativo de AMENITIES que soporta *freeTribe*. Desde esta herramienta se pueden crear, modificar y eliminar cada uno de dichos objetos.

3.4 Segunda fase: evaluación

Teniendo en cuenta que el procedimiento propuesto es iterativo, y que uno de sus objetivos es mostrar las principales propiedades de los frameworks, comencemos a evaluar estos principios en *freeTribe*. Comenzaremos con las propiedades expuestas en el procedimiento y, en caso que se desee se incluirá alguna otra (el procedimiento lo permite).

Alineación y pertenencia al dominio

La instanciación de un framework no es otras cosa que la modificación de sus puntos calientes para ajustarlo a una aplicación específica dentro del dominio de problemas en que éste está enfocado. *freeTribe* implementa de forma acertada los conceptos definidos en AMENETIES, los cuales constituyen los elementos más relevantes dentro del domino CSCW. Además los puntos de extensión, los metacasos de uso, y las abstracciones definidas aseguran que las aplicaciones construidas sobre el framework estarán siempre dentro del dominio CSCW. Las dos aplicaciones de ejemplo mostradas en (**Rodríguez, 06**) sustentan esta afirmación.

Por otra parte, el framework propicia considerablemente a los recursos humanos proporcionar una mayor producción de sistemas colaborativos distribuidos. Entretanto, en el ámbito tecnológico se puede decir que cualquier usuario puede interactuar con el framework sin necesidad de contar con un alto nivel de conocimientos en las diferentes perspectivas del CSCW, ese es uno de los objetivos de *freeTribe*, y se cumplió. *freeTribe* incrementa la productividad de los desarrolladores en la construcción de aplicaciones groupware con una alta calidad.

Principales patrones de diseño

En *freeTribe* se emplearon varios patrones de diseño entre los que resulta oportuno mencionar los siguientes:

Patrón **Singleton** (**Gamma, 95**): asegura que una clase tenga una única instancia, y proporciona un punto global de acceso a ésta. Este patrón está presente en el paquete Servidor en la clase ClsServidor.

Patrón **Template Method** (**Gamma, 95**): define un esqueleto de un algoritmo. Este patrón permite que las subclases redefinan ciertos pasos del algoritmo sin cambiar la estructura del mismo.

freeTribe hace uso de este patrón para definir los meta casos de uso; en los cuales se encierra una funcionalidad global, una plantilla extensible, que permite que las subclases redefinan los pasos del algoritmo necesarios para adaptarlo al sistema

particular. En (**Rodríguez, 06**) se describen los MCU (Meta Casos de Uso) con el siguiente formato:

- Nombre del MCU.
- Descripción.
- Pasos.
- Otras clases involucradas.

Este patrón puede emplearse, además, para implementar la materialización y desmaterialización de los objetos persistentes. Por ejemplo, se puede definir una interfaz (esqueleto, plantilla, contrato) con dos métodos, para materializar y desmaterializar el objeto. Luego cada clase que defina objetos persistentes dará la implementación adecuada a esta interfaz. Ellas son las mejores candidatas para realizar este proceso, ya que representan los datos que van a ser almacenados.

Patrón **Representing Objects as Tables** (**Brown, 96**): ¿cómo mapear un objeto a un registro o a un esquema de una base de datos relacional?

Este patrón se usa en *freeTribe* para tener persistentes todos los objetos generados del modelo conceptual en que se basa, como son actores, capacidades, equipos de trabajo, grupo, etc. además, para el control del mecanismo asíncrono distribuido del envío de mensajes de comunicación entre las aplicaciones clientes y entre éstas y las aplicaciones servidoras, ya que estos mensajes quedarán almacenados hasta que el cliente los solicite. Las clases ClsPersistenciaObjetos, ClsTareaCooperativa, entre otras, implementan los conceptos relacionados con este patrón.

Patrón **Object Identifier** (**Brown, 96**): propone asignar un identificador de objeto, OID por sus siglas en inglés, a cada registro y objeto (o Proxy de un objeto).

En *freeTribe* los registros y los objetos tienen un identificador para relacionar fácilmente los registros con los objetos, y asegurar que no existan duplicados inapropiados.

Patrón **Cache Management** (**Brown, 96**): propone que los mapeadores sean los responsables de administrar sus respectivas cachés. Si se usan diferentes mapeadores

para cada una de las clases de objetos persistentes, éstos deben mantener sus propias cachés.

En *freeTribe* los servicios persistentes almacenan en una caché (un buffer de objetos) los objetos materializados por razones de rendimiento, aumentando así las prestaciones de todo el sistema de persistencia.

En los **Anexos A** y **B** se hace una referencia de los patrones que con más frecuencia se emplean en el desarrollo de los frameworks, por lo que constituye una herramienta de valor para la evaluación de los mismos.

Inclusión del principio "Inversión de Control"

freeTribe muestra esta propiedad al hacer uso del patrón "Template Method", por medio del cual el framework retiene el control del flujo de ejecución de las aplicaciones. El código de las aplicaciones que instancien a *freeTribe* será llamado en los puntos definidos por el framework (puntos calientes), por lo que éste maneja el flujo de ejecución.

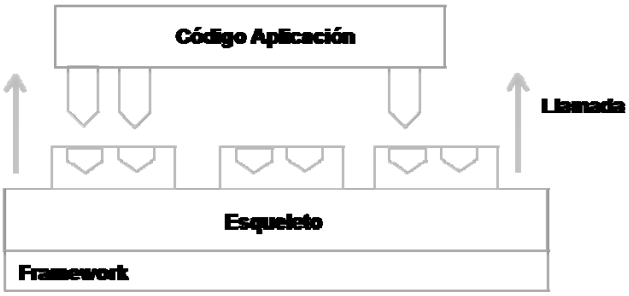

Figura XVI Flujo de ejecución.

Además *freeTribe* implementa los conceptos relacionados a los contenedores y a la búsqueda de dependencias expuestos en los **Capítulos 1** y **2** respectivamente. Esto se observa en los ejemplos proporcionados en (**Rodríguez, 06**):

```
ClsdelegcontenedorcontenedorMensajero;
ClsdelegsessionsesionSesion;
...
contenedorMensajero = servidor.obtenerContenedor("Mensajeria");
sesion = contenedorMensajero.obtenerSessionIdentSinPersistencia("SessionMensajeria");
```

Figura XVII Fragmento de código del chat presentado en (Rodríguez, 06).

Donde primeramente se inicializa el entorno de ejecución de *freeTribe*, en el que se le indica al componente las características del entorno de ejecución (por ejemplo, qué servicios tiene a su disposición, como persistencia, servicios de mensajería, etc.) y continúa con el contenedor llamando a las operaciones implementadas por el componente cada vez que se requiere su ejecución.

En los entornos distribuidos es muy común el uso de este principio y la inversión de control se da a muchos niveles.

Inclusión del principio "Separación de Intereses"

Como se ha explicado en el **Capítulo 2**, existen varias estrategias para implementar este principio. En *freeTribe* se hace un uso adecuado de los patrones de diseño, por lo que se logra cumplir, en una medida aceptable, con esta propiedad. Además, el framework consta de una estructura en paquetes, con una dependencia mínima entre ellos. Esto permite enfocar a los desarrolladores de acuerdo a sus habilidades y conduce a una mayor reutilización.

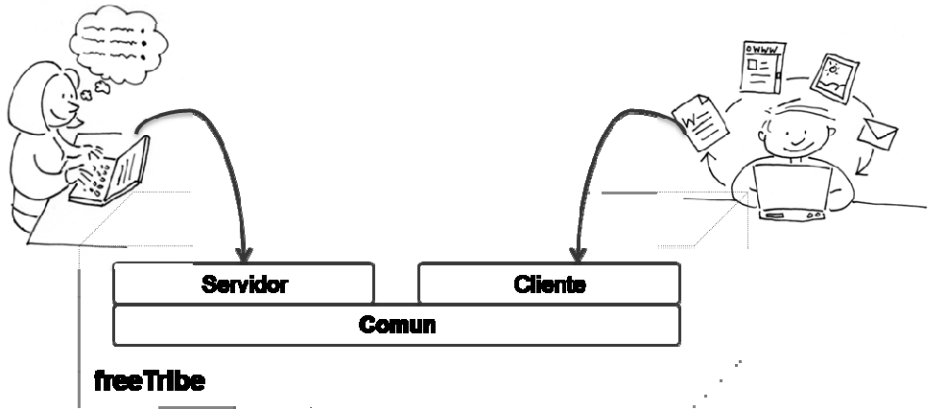

Figura XVIII Separación de Intereses en *freeTribe*.

El framework desarrolla tres aristas o intereses esenciales para los sistemas colaborativos y para todos en general, la seguridad, persistencia y manejo de excepciones. Estos elementos pueden beneficiarse considerablemente de la aplicación de AOP (**Kickzales, 97**). Por lo que se recomienda estudiar la aplicación, en futuras versiones, de los conceptos relacionados con la Programación Orientada a Aspectos (**Kickzales, 97**), usando, tal vez, AspectJ (**Laddad, 03**) - debido a que el framework está desarrollado sobre Java - para la implementación de los conceptos relacionados con la seguridad, la persistencia y el manejo de excepciones.

Por ejemplo, se puede emplear los conceptos de AOP en el manejo de las excepciones, lo que resultaría en algo así:

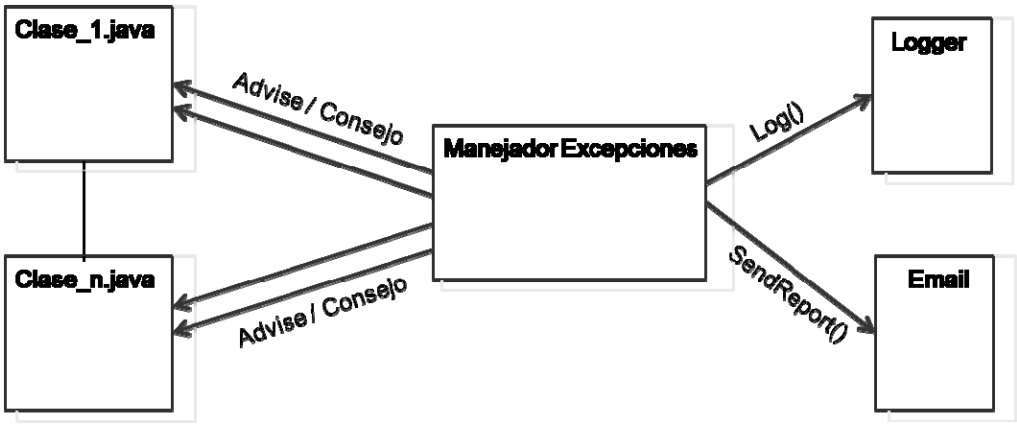

Figura XIX Manejo de Excepciones con AOP.

Muy similar a lo anterior podrían implementarse las directivas de autenticación y autorización y las reglas o contratos del dominio, entre otros conceptos del framework. Ello aseguraría la obtención de una mayor modularidad y aportaría los siguientes beneficios (**Laddad, 03**):

- **Claridad**: código fácil de entender y modificar.
- **Reusabilidad**: la base para la reutilización es la existencia de un bajo acoplamiento (poca dependencia) entre los módulos del sistema. Debido a que AOP implementa cada aspecto por separado, los módulos bases no tienen conciencia de la existencias de los otros, sólo el modulo que contiene las especificaciones de las reglas para tejer el código. Por lo tanto, simplemente cambiando este modulo, se modifica la configuración del sistema completo.
- **Fácil de desarrollar y mantener**: la adición de una nueva funcionalidad ahora es una cuestión de incluir un nuevo aspecto y no requiere ningún cambio a los módulos base. Además, cuando se agrega un nuevo modulo al sistema, los aspectos existentes también pueden capturan sus requerimientos, ayudando a crear una evolución coherente.

Extensibilidad

Los desarrolladores pueden extender el comportamiento del framework, ya sea por medio de la herencia, o por adición de nuevas clases que implementen las interfaces que este provee. *freeTribe* posibilita esto mediante la inclusión de clases abstractas, tales como Contenedor, que pueden ser especializadas de acuerdo a las funcionalidades de la aplicación en desarrollo; o por medio de la implementación de las interfaces, por ejemplo, *autentificacionOperations*, donde queda establecido una especie de contrato, por medio del cual las clases que hagan referencia a ella quedan comprometidas a desarrollar las actividades de autenticación definidas en la interfaz.

3.5 Conclusiones parciales

La pretensión de este capítulo ha sido mostrar cómo es posible abordar mediante el procedimiento propuesto la evaluación de las principales propiedades y/o principios que caracterizan a los frameworks. Además, cómo este proceso es aplicable tanto en las etapas de ingeniería del framework como después de realizado, como una herramienta para su evolución y/o certificación de parámetros de calidad del mismo.

El procedimiento se aplicó a un framework real, *freeTribe*. Framework para el desarrollo de aplicaciones colaborativas distribuidas. Como resultado se obtuvo una serie de recomendaciones y estrategias a tener en cuenta en el desarrollo de futuras versiones del framework. Revelando a los desarrolladores dónde enfocarse para cubrir el amplio espectro de propiedades y características, garantizando así una evolución satisfactoria y dentro de los parámetros de calidad definidos.

El presente capítulo, junto con los resultados de la aplicación del método Delphi para la obtención de un pronóstico confiable de la utilidad práctica real del procedimiento propuesto, constituyen el punto de partida para una validación y aportan, además, elementos de interés para la evolución y mejora del procedimiento.

CONCLUSIONES FINALES

La ejecución de la investigación desarrollada permitió arribar a las siguientes conclusiones.

El desarrollo de frameworks correctamente implementados, reusables, dentro de los parámetros de calidad aceptables y que respondan a las necesidades de los usuarios es realmente una tarea compleja. El éxito en el desarrollo de frameworks depende en buena medida de la atención que le preste el equipo a las propiedades y/o principios que los caracterizan, que si bien no resultan imprescindibles sí son de vital importancia para la calidad y madurez del framework, tales como (**Fayad, 97; Fayad, 00; Markiewicz, 01; Johnson, 04**): *alineación y pertenencia al dominio*, *patrones de diseño*, *extensibilidad*, *inversión de control*, *separación de intereses*, etc.

La investigación desarrollada permitió elaborar una propuesta de procedimiento para la evaluación de los frameworks, que cumple el objetivo de exponer y evaluar los principios y/o propiedades mencionados y resulta aplicable durante el proceso de ingeniería del framework, así como después de realizado, como una herramienta para su evolución y/o certificación de parámetros de calidad.

El procedimiento elaborado parte de un proceso de conceptualización y propone la evaluación, mediante un proceso iterativo, de todas y cada una de las propiedades expuestas, permite la incorporación de nuevos atributos y sugiere, además, una evolución del proceso iterativo que cubre todas estas propiedades.

El procedimiento propuesto está avalado por la opinión de 26 expertos en el tema que fueron encuestados, así como por la aplicación práctica en el framework *freeTribe* (**Capítulo 3)**, lo que resulta un punto de partida importante para la validación del procedimiento, cumpliendo así los objetivos planteados al inicio de la investigación, demostrando la hipótesis supuesta.

RECOMENDACIONES

Partiendo de los resultados de la investigación efectuada y de la experiencia adquirida durante la realización de este trabajo, y con el propósito de asegurar la posterior ampliación, modificación y mejora del procedimiento propuesto, se exponen a continuación algunas líneas de investigación que consideramos importante desarrollar; así como las recomendaciones.

Mantener actualizado el procedimiento propuesto, incorporándole las recomendaciones realizadas por los expertos y extender su aplicación a un número más amplio de casos reales.

Desarrollar una herramienta que permita automatizar el procedimiento propuesto en la medida de lo posible.

Presentar el procedimiento propuesto en próximos eventos científicos de la UCI u otras instancias.

Profundizar en el estudio de metodologías para la Ingeniería de Software aplicada a los frameworks, enfatizando en temas de importancia medular para su construcción, evolución y documentación satisfactoria, tales como, métricas para la estimación del costo, atributos de calidad, evaluación de arquitecturas, etc.

BIBLIOGRAFÍA

(Alexander, 77) **Alexander, Christopher**. *A Pattern Language.* Nueva York, Oxford University Press, 1977.

(American, 04) The Editors of the American Heritage Dictionaries. The American Heritage Dictionary of the English Language. Fourth Edition. The American Heritage Dictionaries, 2006.

(Arango, 91) **Arango, G. and R. Prieto-Diaz**. *Domain Analysis Concepts and Research Directions*, IEEE Computer Society, 1991.

(Archer, 01) **Archer Tom**. A fondo C#. McGraw Hill Profesional, 2001.

(Beck, 99) **Beck, Kent**. *Extreme Programming Explained*. 1999.

(Booch, 99) **Booch, Grady, James Rumbaugh and Ivar Jacobson**. *El Lenguaje Unificado de Modelado*. Addison Wesley Iberoamericana, 1999.

(Brown, 96) **Brown, Kyle and Bruce Whitenack**. *Crossing Chasms. Pattern Languages of Program Design*. Addison Wesley, 1996.

(Bushmann, 96) **Bushmann, Frank, Regine Meunier, Hans Rohnert*, et al.*** *Pattern-Oriented Software Architecture: A System of Patterns*. England, John Wiley and Sons, 1996.

(D´Souza, 99) **D´Souza, Desmond Francis**. *Objects, Components and frameworks with UML. The Catalysis Approach*. Addison Wesley, 1999.

(Escofet, 04) **Escofet, Eduardo**. *Diseño de una Aplicación Groupware de Consultas*. España, Universidad de Granada, 2004.

(Fayad, 97) **Fayad, Mohamed E. and David C. Schmidt**. *Object-Oriented Application frameworks*. Communications of the ACM, 1997.

(Fayad, 99) **Fayad, Mohamed E., David C. Schmidt and Ralph E. Johnson**. *Building Application frameworks*. Addison Wesley, 1999.

(Fayad, 00) **Fayad, Mohamed E. and David S. Hamu**. *Enterprise frameworks: guidelines for Selection*. The Communications of ACM, 2000.

(Fugini, 92) **Fugini, Mariagrazia, Oscar Nierstrasz and Barbara Pernici**. *Application Development Through Reuse: The ITHACA Tools Environment*, 1992.

(Gamma, 95) **Gamma, Erich, Richard Helm, Ralph Johnson*, et al.*** *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.

(Garrido, 03) **Garrido, José Luis**. *Una Metodología para el desarrollo de sistemas cooperativos basada en modelos de comportamiento y tareas*. Lenguajes y Sistemas Informáticos. Granada, Universidad de Granada, 2003.

(Johnson, 88) **Johnson, Ralph E. and Brian Foote**. Designing Reusable Classes. *Journal of Object-Oriented Programming*, 1988.

(Johnson, 97) **Johnson, Ralph E**. *Components, Frameworks, Patterns*. Proceedings of the 1997 symposium on Software reusability, 1997.

(Johnson, 01) **Johnson, Ralph E**. *Documenting Frameworks Using Patterns.* OOPSLA, New York, ACM Press, 2001.

(Johnson, 04) **Johnson, Rod and Juergen Hoeller**. *Expert One-on-One J2EE Development without EJB*. Wiley Publishing, Inc, 2004.

(Kaisler, 05) **Kaisler, Stephen H.** *Software Paradigms*. New Jersey, John Wiley & Sons, Inc, 2005.

(Kang, 90) **Kang, K., S. Cohen, J. Hess*, et al.*** *Feature-Oriented Domain Analysis (FODA). Feasibility Study*. Pittsburgh, Software Engineering Institute, 1990.

(Kickzales, 97) **Kickzales, Gregor, John Lamping, Anurag Mendhekar*, et al.*** *Aspect - Oriented Programming*. European Conference on Object-Oriented Programming (ECOOP), Finlandia, 1997.

(Kruchten, 00) **Kruchten, Philippe**. *The Rational Unified Process. An Introduction*. Second Edition. Addison Wesley, 2000.

(Kruchten, 03) **Kruchten, Philippe**. *The 4+1 View Model of Architecture*. IEEE Software, 1995.

(Laddad, 03) **Laddad, Ramnivas**. *AspectJ in Action. Practical Aspect-Oriented Programming*. Manning Publications Co, 2003.

(Landeta, 99) **Landeta, Jon.** El método Delphi. Una técnica de previsión del futuro. Ariel, 1999.

(Larman, 99) **Larman, Craig**. *UML y Patrones. Introducción al análisis y diseño orientado a objetos*. Prentice Hall, 1999.

(Larman, 04) **Larman, Craig**. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. 3ra ed. Addison Wesley Professional, 2004.

(Marchesi, 02) **Marchesi, Michele, Giancarlo Succi, Don Wells*, et al.*** *Extreme Programming Perspectives*. Addison Wesley, 2002.

(Markiewicz, 01) **Markiewicz, Marcus E. and Carlos Lucena**. *Object Oriented Framework Development*, 2001.

(Opdyke, 90) Opdyke W.F., R.E. Johnson. *Refactoring: An aid in designing object-oriented application frameworks*. Symposium on Object-Oriented Programming Emphasizing Practical Applications, 1990.

(Opdyke, 92) Opdyke W.F. *Refactoring Object-Oriented Frameworks*. University of Illinois at Urbana-Champaign, 1992.

(Quintero, 00) **Quintero, Antonia Reina**. *Visión General de la Programación Orientada a Aspectos*. Dpto. de Lenguajes y Sistemas Informáticos. Sevilla, Universidad de Sevilla, 2000.

(Riehle, 00) **Riehle, Dirk**. *Framework Design. A Role Modeling Approach*. Hamburg, University of Hamburg, 2000.

(Rodríguez, 06) **Rodríguez, Julio and Lázaro Hurtado**. *Propuesta de un framework para sistemas colaborativos distribuidos.* Holguín, Universidad de Holguín "Oscar Lucero Moya", 2006.

(Rosenberg, 05) **Rosenberg, Doug, Matt Stephens and Mark Collins-Cope**. *Agile development with ICONIX process: people, process, and pragmatism*. Apress, 2005.

(Szyperski, 98) **Szyperski, Clemens**. *Component Software-Beyond Object-Oriented Programming*. Massa-chusetts, Addison Wesley, 1998.

(Taligent, 93) **Taligent, Corp**. *Leveraging Object-Oriented frameworks*. Cupertino, CA, 1993.

(Taligent, 94) **Taligent, Corp**. *Building Object-Oriented frameworks*. Cupertino, CA, 1994.

(Taligent, 95) **Taligent, Corp**. *The Power of frameworks*. Addison Wesley, 1995.

ANEXOS

Anexo A: Principales patrones aplicados en el desarrollo de frameworks (**Gamma, 95; Brown, 96**).

- **Adapter**.

  Convierte la interfaz de una clase para que se adapte a lo que el cliente que la usa necesita, permitiendo así que trabajen juntas clases cuyas interfaces son incompatibles.

- **Proxy**.

  Proporciona un representante o delegado que se encargado de controlar el acceso a un objeto, generalmente por motivos de eficiencia.

- **Chain of Responsibility**.

  Proporciona a más de un objeto la capacidad de atender una petición, para así evitar el acoplamiento. Se forma con estos objetos una cadena, en la cual cada objeto o satisface la petición o la pasa al siguiente.

- **Composite**.

  Permite a los clientes tratar uniformemente a los objetos simples y compuestos de una estructura jerárquica recursiva.

- **Builder**.

  Separa la construcción y la representación de un objeto complejo, para así permitir que el mismo proceso de construcción sirva para crear diferentes representaciones.

- **Decorator**.

  Añade responsabilidades adicionales a un objeto dinámicamente, proporcionando una alternativa flexible a la especialización mediante herencia, cuando se trata de añadir funcionalidades.

- **State**.

  Permite a un objeto modificar su comportamiento a medida que su estado interno va cambiando, dando así la impresión de que el objeto “cambia de clase”.

- **Strategy**.

  Define una familia de algoritmos encapsulando por separado cada uno de ellos y haciéndolos, por tanto, intercambiables. Esto permite a los algoritmos variar con independencia de los clientes que los usan.

- **Abstract Factory**.

  Proporciona una interfaz para la creación de familias de objetos interdependientes o interrelacionados, sin especificar sus clases concretas.

- **Facade**.

  Proporciona una interfaz unificada de alto nivel que, representando a todo un subsistema, facilite su uso. La “fachada” satisface a la mayoría de los clientes, sin ocultar las funciones de menor nivel a aquellos que necesiten acceder a ellas.

- **Interpreter**.

  Dado un lenguaje, define una representación para su gramática junto con un intérprete que usa dicha representación para interpretar sentencias en ese lenguaje.

- **Iterator**.

  Proporcionar una forma de acceder secuencialmente a los elementos de un objeto compuesto por agregación sin necesidad de desvelar su representación interna.

- **Factory Method**.

  Define una interfaz para la creación de un objeto, pero permitiendo a las subclases decidir de qué clase instanciarlo. Permite, por tanto, que una clase difiera la instanciación en favor de sus subclases.

- **Template Method**.

  Define el esqueleto de un algoritmo para una operación, dejando para sus subclases la capacidad de redefinir el funcionamiento de los pasos de este algoritmo, siempre y cuando la estructura del mismo permanezca intacta.

- **Observer**.

  Define una dependencia “uno a muchos” entre objetos, para que, cuando uno de ellos cambie su estado, todos los que dependan de él sean avisados y puedan actualizarse convenientemente.

- **Prototype**.

  Especifica los tipos de objetos que queremos crear usando una instancia prototípica y crear nuevos objetos copiando dicho prototipo.

- **Singleton**.

  Garantiza que una clase sólo tiene una única instancia, proporcionando un punto de acceso global a la misma.

- **Representing Objects as Tables.**

  ¿Cómo mapear un objeto a un registro o a un esquema de una base de datos relacional?

- **Object Identifier.**

  Propone asignar un identificador de objeto, OID por sus siglas en inglés, a cada registro y objeto (o Proxy de un objeto)

- **Cache Management.**

  Propone que los mapeadores sean los responsables de administrar sus respectivas cachés. Si se usan diferentes mapeadores para cada una de las clases de objetos persistentes, éstos deben mantener sus propias cachés.

Anexo B: Encuesta a expertos.

ENCUESTA A EXPERTOS.

Nombre y apellidos: _____.

Institución a la que pertenece: _____.

Cargo actual: _____.

Calificación profesional, grado científico o académico:

Profesor: _____.

Licenciado: _____.

Especialista: _____.

Máster: _____.

Doctor: _____.

Años de experiencia en el cargo: _____.

Años de experiencia docente y/o en la investigación: _____.

Con el propósito de sustentar la validez del procedimiento propuesto en el trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas, dirigido al diagnostico y evaluación de las propiedades y/o principios de los frameworks, el cual se adjunta a la presente encuesta, se requiere conocer sus apreciables opiniones y sugerencias referentes a:

- Grado de relevancia de las fases y actividades del procedimiento.
- Grado de relevancia de las propiedades básicas incluidas en el procedimiento.
- ¿Qué otras propiedades pueden incluirse o eliminarse de la propuesta de procedimiento de evaluación de los frameworks?
- Sugerencias de cambios al procedimiento propuesto, así como a la evolución del proceso iterativo del mismo.

Indicaciones:

Marque con una X la respuesta Usted considera correcta.

1. Especifique a continuación el grado de relevancia que Usted considera tienen las actividades propuestas en el procedimiento.

   Escala:

   MR: Muy relevante. **BR**: Bastante relevante. **R**: Relevante.

   PR: Poco relevante. **NR**: No relevante.

Actividades	**MR**	**BR**	**R**	**PR**	**NR**
Describir Framework					
Analizar Dominio					
Selección de Propiedades					
Caracterización y valoración de propiedades					
Presentar resultados de la evaluación					

2. Especifique a continuación el grado de relevancia que Usted considera tienen las propiedades expuestas en el procedimiento.

   Escala:

   MR: Muy relevante. **BR**: Bastante relevante. **R**: Relevante.

   PR: Poco relevante. **NR**: No relevante.

Propiedades	**MR**	**BR**	**R**	**PR**	**NR**
Alineación y pertenencia al dominio					
Patrones de diseño					
Inversión de Control					
Separación de Intereses					
Extensibilidad					

3. ¿Qué otras propiedades pueden incluirse o eliminarse de la propuesta de procedimiento de evaluación de los frameworks?

Propiedades a incluir	**Propiedades a eliminar**

4. Sugerencias de cambios al procedimiento propuesto, así como a la evolución del proceso iterativo del mismo.

Anexo C: Resultados generales de la encuesta.

SOBRE EL GRADO DE RELEVANCIA DE LAS FASES Y ACTIVIDADES						
TABLA DE FRECUENCIA ABSOLUTA						
	MR	BR	R	PR	NR	TOTAL
Describir Framework	18	4	4	0	0	26
Analizar Dominio	25	1	0	0	0	26
Selección de Propiedades	20	4	2	0	0	26
Caracterización y valoración de propied	24	2	0	0	0	26
Presentar resultados de la evaluación	25	1	0	0	0	26

Figura XX Resultado general, pregunta 1.

SOBRE EL GRADO DE RELEVANCIA DE LAS PROPIEDADES Y/O PRINCIPIOS						
TABLA DE FRECUENCIA ABSOLUTA						
	MR	BR	R	PR	NR	TOTAL
Alineación y pertenencia al dominio	1	1	1	23	0	26
Patrones de diseño	26	0	0	0	0	26
Inversión de Control	22	2	2	0	0	26
Separación de Intereses	13	1	10	1	1	26
Extensibilidad	24	1	1	0	0	26

Figura XXI Resultado general, pregunta 2.

Anexo D: Modelo de componentes de *freeTribe*.

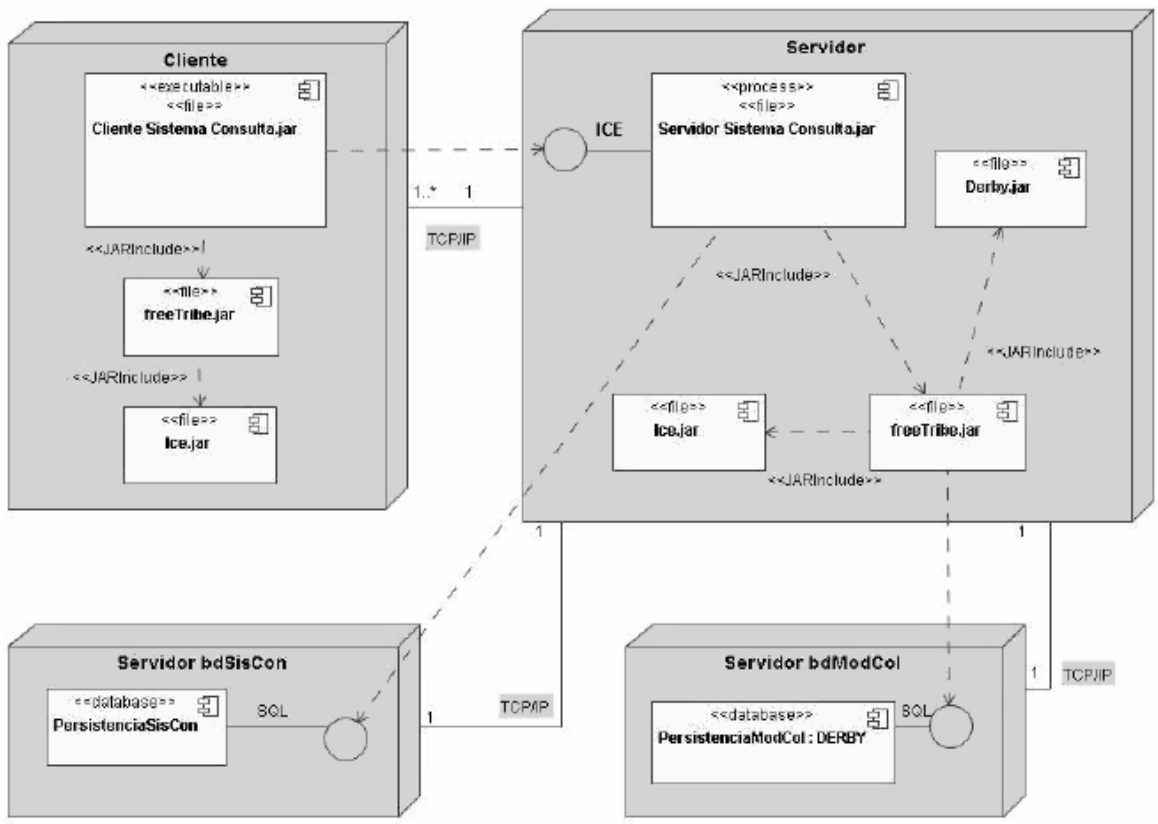

Figura XXII Modelo de componentes de *freeTribe*.

GLOSARIO

AMENETIES

Acrónimo para "**A ME**thodology for a**N**alysis and des**I**gn of coopera**TI**ve systems", (**Garrido, 03**). Es una metodología para el desarrollo de sistemas cooperativos basados en modelos de comportamiento y tareas.

AOP

Término acuñado por Gregor Kickzales (**Kickzales, 97**) y define un nuevo paradigma, la "Programación Orientada a Aspectos". La idea básica que persigue AOP es permitir que un programa sea construido describiendo cada concepto separadamente.

API

Es una especificación, escrita en un lenguaje de programación, de las características de un modulo, del cual los clientes pueden depender (**Takeshita, 1997**).

AspectJ

AspectJ es una extensión de uso general, para el trabajo con la Programación Orientada a Aspectos, al lenguaje de programación de Java (**Laddad, 03**).

CSCW

Siglas en inglés de Computer Supported Cooperative Work, Trabajo Cooperativo Asistido por Computadora. Disciplina que tiene como principal objetivo disminuir el distanciamiento entre los aspectos técnicos y sociales dentro del trabajo en grupo (**Beaudouin, 1999**).

freeTribe

Framework para el desarrollo de sistemas colaborativos distribuidos (**FR**amework for d**E**v**E**lopment of dis**TRIB**uted groupwar**E**) (**Rodríguez, 06**).

freeTribeBoss

Herramienta que permite automatizar el proceso de creación de un servidor colaborativo al brindar la posibilidad de definir los conceptos del modelo conceptual del modelo cooperativo de AMENITIES que soporta *freeTribe*. Desde esta herramienta se pueden crear, modificar y eliminar cada uno de dichos objetos (**Rodríguez, 06**).

Frozen Spots

Puntos inmutables que constituyen el núcleo o kernel de un framework (**Markiewicz, 01**).

Hot Spots

Clases o métodos abstractos que deben ser implementados o puestos en ejecución para instanciar un framework (**Markiewicz, 01**).

ICONIX

Se ubica entre el gran tamaño y complejidad de RUP (Rational Unified Process – Proceso Unificado de Rational) y lo pequeño y compacto de XP (eXtreme Programming) (**Escofet, 04**). Al igual que RUP, ICONIX es una metodología conducida por casos de uso, pero no incorpora tantos artefactos UML. Es una metodología relativamente pequeña al igual que XP, pero no descarta las etapas de análisis y diseño. Utiliza UML en sus etapas, de modo que se pueden seguir los requerimientos y adaptarse a nuevos cambios.

XML

Lenguaje Extensible de Marcas, la versión más simple del estándar SGML para la creación y diseño de documentos HTML.

XP

Es una de las metodologías de desarrollo de software más exitosa en la actualidad, utilizada para proyectos de corto plazo y con equipos pequeños. La metodología consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto (**Beck, 99**).