Análisis de Conceptos Formales como soporte para la construcción de Frameworks de dominio*

Félix Prieto, Yania Crespo, José Manuel Marqués, y Miguel Ángel Laguna

Dpto. de Informática. Unviersidad de Valladolid {felix,yania,jmmc,mlaguna}@infor.uva.es

Resumen El desarrollo de un buen framework es un proceso difícil. Esta dificultad es mayor en los frameworks de dominio, que deben adaptarse rápidamente a los cambiantes requisitos de las áreas de negocio que modelan.
En este trabajo mostraremos un proceso de desarrollo especialmente diseñado para este tipo de frameworks, que, gracias al soporte formal que proporcionan una serie de técnicas basadas en el Análisis de Conceptos Formales, y al conjunto de herramientas que lo implementan, proporciona un soporte automático para la construcción de frameworks de dominio
Partiendo de nuestros trabajos previos sobre el proceso de construcción de frameworks de dominio, proponemos la generalización de las técnicas utilizadas a todo el ciclo de vida del framework. Para ello se analizan con detalle las distintas fases del proceso de construcción propuesto, haciendo especial énfasis en los aspectos automatizables del mismo. Además se presenta una herramienta en fase experimental y el uso que de ella se hace para la aplicación de una de las técnicas mostradas al análisis de una biblioteca de clases real, como un pequeño caso de estudio.

1 Introducción

La reutilización ha sido uno de los objetivos de la Ingeniería del Software desde sus orígenes, y mejorar las posibilidades de reutilización fue uno de los acicates para la creación de las técnicas orientadas a objetos. No obstante, y a pesar de las evidentes ventajas que aporta en este sentido, parece ya claro que la clase constituye un elemento reutilizable (asset) de grado demasiado fino como para garantizar buenos niveles de reutilización [21].

Para solventar esta dificultad se han propuesto técnicas de desarrollo basadas en assets de grano más grueso, como los frameworks [10], que han proporcionado éxitos notables, especialmente en el ámbito de las interfaces gráficas de usuario, donde estas técnicas tuvieron origen.

El framework puede ser definido como un conjunto de clases, generalmente algunas de ellas abstractas, y las colaboraciones que se establecen entre ellas, para proporcionar un diseño abstracto de las soluciones de un conjunto de problemas [10,11].

El framework captura las decisiones de diseño comunes a un tipo de aplicaciones, estableciendo un modelo común a todas ellas, asignando responsabilidades y estableciendo colaboraciones entre las clases que forman el modelo. Además, este modelo común contiene puntos de variabilidad, conocidos como puntos calientes[1] [12], capaces de albergar los distintos comportamientos de las aplicaciones representadas por el framework.

La reutilización se produce entonces en el proceso de instanciación del framework, en el que el desarrollador con reutilización proporciona la funcionalidad específica de su aplicación rellenando los puntos calientes.

Aunque se han propuesto diferentes clasificaciones para caracterizar los tipos de frameworks [3,10], todas parecen coincidir en que se pueden distinguir dos tipos denominados

* Este trabajo ha sido parcialmente financiado por la CICYT(TIC2000-1673-C06-05) como parte del proyecto DOLMEN

[1] Puntos calientes, del inglés *hot spots*, también denominados *hooks*.

frameworks de aplicación y *frameworks de dominio*. Un framework de aplicación encapsula una capa de funcionalidad horizontal que puede ser aplicada en la construcción de una gran variedad de programas. Además de los frameworks que implementan las interfaces gráficas de usuario, que representan su paradigma, podemos clasificar en esta categoría los dedicados al establecimiento de comunicaciones o procesamiento de documentos XML entre otros. Por otra parte, un framework de dominio implementa una capa de funcionalidad vertical, correspondiéndose con un dominio de aplicación o una línea de producto. Estos deberán ser los más numerosos, y su evolución deberá ser también la más rápida, pues deben adaptarse a las áreas de negocio para las que están diseñados.

También la forma en que se instancian sus puntos calientes permite clasificar los frameworks [10]. Así hablamos de puntos calientes, y por extensión de frameworks, de *caja blanca* cuando su instanciación se realiza mediante técnicas de herencia, y por ello requiere de conocimientos sobre la implementación del framework por parte del desarrollador.

La categoría de *caja negra* se refiere a puntos calientes y frameworks en que la instanciación se realiza mediante composición e instanciación de parámetros genéricos. Su utilización es mucho más simple por cuanto sólo requiere seleccionar de entre un conjunto de opciones predefinidas. Sin embargo, y por los mismos motivos su construcción resulta mucho más difícil.

A pesar de sus esperados beneficios, algunos obstáculos han impedido implantar, de manera exitosa y generalizada, modelos de reutilización basados en frameworks. Las principales dificultades han venido dadas porque son difíciles de construir y no es sencillo aprender a utilizarlos. Prueba de ello es el buen número de trabajos dedicados a informar sobre la experiencia obtenida en la construcción y utilización de frameworks, como se puso de manifiesto en el "Electronic Symposium on Object-Oriented Application Frameworks" [4].

Para comprender el origen de estas dificultades hay que tener en cuenta que la construcción de frameworks es un proceso puramente artesanal. Ello conduce a un desarrollo que requiere de grandes dosis de experiencia, es costoso y propenso a errores. Por ello, para generalizar la utilización de frameworks de dominio es necesario adoptar un enfoque más industrial, basado en estrategias y herramientas capaces de minimizar la inversión inicial necesaria y reducir los costes asociados al mantenimiento del framework.

En este sentido en [15] propusimos una estrategia de construcción de frameworks de dominio que parte de la elaboración previa de un conjunto de aplicaciones del dominio. A diferencia de otras propuestas que utilizan estas aplicaciones como punto de partida para un proceso de generalización manual, o simplemente como medio informal de adquirir experiencia en el dominio en cuestión, nuestra apuesta pasa por extraer los aspectos comunes a todas estas aplicaciones mediante herramientas automáticas basadas en técnicas formales. Esta primera versión del framework debe ser considerada como un producto intermedio, aunque ya utilizable, de un proceso más amplio que, también con la asistencia de herramientas adecuadas, debe contemplar el proceso completo de desarrollo del framework, con especial énfasis en su adaptación para responder a los cambiantes requisitos de las áreas de negocio en que estos artefactos software son finalmente utilizados.

En el presente trabajo propondremos un proceso de construcción de frameworks de dominio soportado por herramientas, que integra la estrategia anteriormente propuesta con las técnicas que facilitan la posterior evolución del framework, tanto para mejorar su capacidad de instanciar aplicaciones del dominio como para adaptarse a las variaciones que indudablemente han de producirse en los requisitos del mismo.

En lo siguiente este trabajo se organiza de la siguiente forma: Dedicamos la sección 2 a describir el proceso de construcción de Frameworks de Dominio en cuestión, enmarcándolo en las propuestas de construcción evolutiva de Frameworks. La sección 3 repasa las técnicas que proporcionan el soporte formal para las herramientas que proporcionan asistencia al proceso anterior, y la sección 4 presenta la arquitectura propuesta para este conjunto de herramientas y la implementación experimental que de una parte de ellas se ha realizado,

junto con los datos relativos a un pequeño caso de estudio realizado con una biblioteca de clases real. Para terminar, presentamos algunas conclusiones y propuestas de trabajo futuro.

2 Un proceso de desarrollo de frameworks de dominio

Los frameworks son aceptados como un asset de grano adecuado para fomentar el proceso de reutilización. Sin embargo, como hemos indicado, su éxito en la práctica ha sido bastante limitado fuera del ámbito de los frameworks de aplicación en que estas técnicas se originaron, y ello a pesar de que en estos días son escasos los sistemas completamente construidos desde cero.

En realidad esto no debería sorprendernos. Si diseñar un sistema es costoso, diseñar un sistema general reutilizable lo es mucho más. Un framework debe encerrar una teoría del dominio del problema y debería ser siempre el resultado de un análisis de dominio, sea explícito u oculto, formal o informal. El diseño de un sistema que cumpla sus requisitos y además encierre la solución para un amplio rango de problemas futuros, es un verdadero reto.

Debido a su coste y complejidad de desarrollo, los frameworks deberán ser construidos solamente cuando se advierte que muchas aplicaciones serán desarrolladas en un dominio específico, permitiendo que la inversión realizada en el desarrollo del framework se amortice al reutilizarlo. En resumen, por todas las razones anteriores, se estima que las situaciones en que es económicamente rentable afrontar la construcción de un framework son aquellas en que se dispone de varias aplicaciones del mismo dominio (o se deben producir en breve plazo) y se espera que se requieran nuevas aplicaciones con cierta frecuencia.

También se admite como cierto que el framework no puede ser considerado como un producto final, sino que debe ser esencialmente evolutivo. Por esa razón se han propuesto varias estrategias para la fabricación de frameworks [16,17] que parten del desarrollo de una o varias aplicaciones del dominio, a partir de las cuales se inicia la construcción de un primer framework. Posteriormente, con la información proporcionada por las sucesivas instanciaciones, dicho framework va refinándose y transformándose.

Sin embargo con este enfoque, la información contenida en las aplicaciones ya desarrolladas se utiliza de un modo informal, como una forma de adquirir experiencia y conocimientos del dominio, o simplemente como un punto de partida para el desarrollo. Esto supone que buena parte de esa información puede ser utilizada de un modo insuficiente en un proceso que, en todo caso, requiere de buenas dosis de habilidad y experiencia por parte del desarrollador, además de ser propenso a errores.

Por otra parte, tampoco con este enfoque se da un soporte adecuado para el posterior proceso de evolución de los frameworks de dominio. Los frameworks comparten con el resto de los artefactos software la necesidad de evolucionar para adaptarse a los cambios de la realidad que pretenden modelar. En este sentido las mismas técnicas aplicables al resto del software son susceptibles de utilización en el caso de los frameworks.

Sin embargo en el caso de los frameworks surgen necesidades específicas relacionadas con la característica distintiva de éstos: su capacidad para recoger el modelo común a un conjunto de aplicaciones que pretenden dar solución a un conjunto de problemas relacionados. Es preciso entonces abordar la necesidad de un framework de evolucionar conforme al modelo de construcción evolutiva, ampliamente admitido en la actualidad, para mejorar su capacidad de ser instanciado en aplicaciones, y esta evolución debe ser planteada en dos dimensiones distintas:

- Necesidad de ampliar la capacidad de instanciación del framework.
- Necesidad de facilitar las instanciaciones que ya son posibles.

La primera versión del framework no es sino un primer diseño que debe ser ampliamente mejorado para ir adaptándose a las nuevas necesidades detectadas durante su utilización, ya

sea gracias a la introducción de nuevos puntos calientes o la reubicación de los ya existentes. Este tipo de mejoras no se restringen a las primeras fases del desarrollo del framework, sino que deben aplicarse siempre que se detecten nuevas necesidades que deben ser cubiertas por el mismo.

Conforme se detecta que el diseño del dominio recogido en el framework va estabilizándose, el interés se desplaza a la forma en que los puntos calientes han sido implementados, puesto que los conocimientos adquiridos en el desarrollo permiten ya establecer de un modo más preciso cuáles son las diferentes implementaciones necesarias para ese punto caliente. Esta información permite abordar la refactorización [2] del mismo para, mediante técnicas de caja negra, posibilitar la instanciación simplemente mediante la elección entre un conjunto de opciones predefinidas.

A partir de estas ideas, proponemos la definición de un proceso de construcción que integre el modelo informal de construcción evolutiva a partir de aplicaciones iniciales del dominio con la utilización de herramientas basadas en técnicas formales de representación del conocimiento. De esta forma pretendemos garantizar la utilización de la información proporcionada por las aplicaciones de una forma intensiva.

Nuestra apuesta pasa por la utilización de herramientas basadas en el Análisis de Conceptos Formales(ACF), una técnica formal de representación del conocimiento que permite poner de manifiesto de forma automática la estructura subyacente en un conjunto de datos.

2.1 El proceso propuesto

En este apartado se presenta una descripción del proceso propuesto. Nuestra propuesta consiste en partir de un proceso de desarrollo de Frameworks de Dominio como los que se describen en [16] o [17], introduciendo en la fase de construcción de la primera versión y en las posteriores fases de evolución, el uso de las técnicas basadas en el ACF para mecanizar la utilización de la información contenida en las aplicaciones iniciales y las instancias del framework.

En una primera fase, las técnicas basadas en el ACF facilitan la detección de los aspectos comunes a un conjunto de aplicaciones iniciales del dominio, aspectos que deberemos recoger en el núcleo del framework, y separarlos de los elementos específicos de cada aplicación concreta. Estos elementos específicos dan lugar a los puntos de variabilidad que serán implementados mediante los puntos calientes. A esta forma de utilizar el ACF, propuesta inicialmente en [8], le denominamos Análisis de Herencia.

En esta misma fase se aplica también otro tipo de análisis, que denominaremos Análisis de Cliente, basado en el ACF, propuesto inicialmente en [19]. Este análisis permite obtener una mejor composición de las clases mediante el reagrupamiento de sus métodos.

Para la posterior fase de evolución del framework apostamos por utilizar la información disponible en las instancias del framework mediante herramientas y técnicas capaces de extraer los datos relevantes para guiar estos procesos de evolución. Las dos dimensiones de evolución que hemos mencionado anteriormente corresponderán con dos subprocesos diferenciados, y dos tipos de herramientas con objetivos distintos.

La necesidad de ampliar la capacidad de instanciación del framework se detecta cuando, al intentar instanciar una nueva aplicación, el modelo no presenta puntos de variabilidad suficientes para ello. La estrategia general consiste entonces en continuar la construcción de la aplicación, a partir de ese punto mediante técnicas convencionales, y aplicar posteriormente las técnicas basadas en ACF para analizar las relaciones de herencia y cliente.

[2] Refactorización, del inglés *Refactoring*, es una forma especial de transformación de software Orientado a Objetos que se caracteriza por no depender de la semántica del software a modificar, tener como objetivo refinar diseños y ser realizada mediante reestructuración y reorganización de clases y agrupaciones.

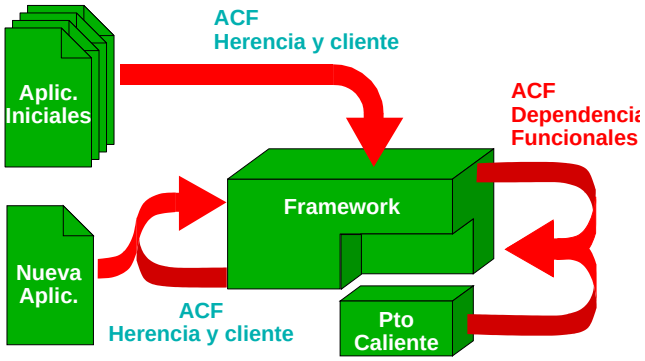

Fig. 1. Diagrama del proceso de construcción completo.

Con ello obtendremos nuevos puntos calientes, implementados con técnicas de caja blanca, que proporcionarán los nuevos puntos de variabilidad requeridos. De los nuevos puntos calientes dispondremos entonces de dos instancias distintas, la que proporciona la nueva aplicación, y la correspondiente a las clases del framework que han salido de su núcleo.

El segundo tipo de proceso no debe ser disparado por una necesidad concreta, sino por la disponibilidad de un buen número de instancias del framework. Cuando la cantidad de instancias haga suponer que disponemos de representantes de un suficiente número de opciones de instanciación, podremos abordar su transformación hacia la tecnología de caja negra.

Un enfoque inocente podría suponer que basta con introducir las clases que instancian el punto caliente dentro del núcleo del framework como alternativas que deben ser seleccionadas mediante técnicas de composición, pero esa estrategia no es en absoluto suficiente.

Los puntos calientes de caja blanca pueden agrupar, y de hecho así suele ocurrir, varias dimensiones de variabilidad. Esto no es un problema, puesto que el desarrollador con reutilización debe comprender el punto caliente antes de instanciar su aplicación. Sin embargo es imprescindible que los puntos calientes de caja negra contengan una única dimensión de variabilidad [13], para que sus combinaciones se realicen en el desarrollo con reutilización, permitiendo así una mayor libertad en el proceso de instanciación y un menor número de opciones predefinidas.

Necesitamos, por lo tanto, herramientas y técnicas capaces de detectar los patrones de variación de las instancias del punto caliente, patrones que permitirán inferir qué métodos del punto caliente pueden ser agrupados en una dimensión de variabilidad. Para esto se utiliza también el ACF mediante una técnica que denominamos Análisis de Dependencias Funcionales.

De este modo podemos proponer la división del punto caliente en diferentes dimensiones de variabilidad, lo que redundará en una mayor facilidad para la posterior implementación mediante técnicas de caja negra.

En resumen, el proceso completo para el desarrollo de Frameworks de Dominio, descrito esquemáticamente en la Figura 1, se inicia a partir de un conjunto de aplicaciones del dominio sobre las que se aplican herramientas basadas en el ACF que analizan tanto las relaciones de herencia como las de cliente. Como consecuencia se obtienen las partes comunes de estas aplicaciones, que pasarán a formar el núcleo de la primera versión del framework, y los puntos de variabilidad que permitirán instanciar las distintas aplicaciones.

Esta primera versión del framework debe evolucionar en función de las necesidades detectadas en el posterior proceso de instanciación.

Cuando en una de estas instanciaciones se descubran requisitos de variabilidad no contemplados en el framework deberemos proceder a construir nuevas aplicaciones que implementen los requisitos solicitados y a unificarlas nuevamente con el framework disponible, mediante

técnicas basadas en herencia y cliente, dando lugar a nuevos puntos calientes que dotan al framework de la variabilidad deseada.

Cuando el número de instancias disponibles de alguno de los puntos calientes del framework haga sospechar que todas sus opciones de variabilidad están ya implementadas, deberemos proceder a su división en dimensiones independientes de variabilidad, aplicando en este caso las técnicas basadas en el análisis de dependencias funcionales.

3 Soporte teórico para el proceso

El Análisis de Conceptos Formales fue introducido por Wille en [20] y aparece totalmente desarrollado en [5]. Se trata de una técnica matemática que permite poner de manifiesto las abstracciones subyacentes en una tabla de datos, formalmente un contexto, mediante la construcción de un retículo de conceptos, también conocido como retículo de Galois, asociado a ésta. El ACF ha sido utilizado en trabajos relacionados con Representación del Conocimiento [9] y con Ingeniería del Software [18].

La técnica básica en ACF consiste en la elaboración de una matriz de incidencia, denominada contexto formal en el lenguaje de la teoría, a partir de la cual se construye de forma algorítmica un retículo de Galois, representable mediante su correspondiente diagrama de Hasse, que contiene toda la información original, si bien organizada de un modo que pone de manifiesto la estructura de los datos.

La aplicación de esta herramienta formal requiere por tanto definir la forma en que será construida la matriz de incidencia, y la forma en que el retículo debe ser posteriormente interpretado. Diferentes matrices de incidencia permiten poner de manifiesto diferentes estructuras en los datos originales. En esta sección ilustraremos de manera informal el modo en que esta técnica permite realizar los análisis de herencia, cliente y dependencias funcionales mencionados en la sección anterior.

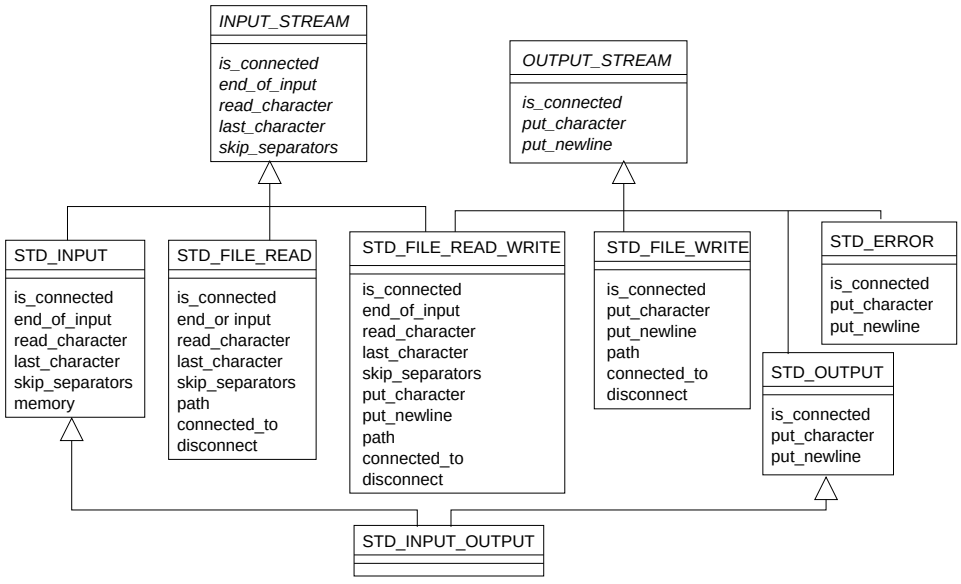

Fig. 2. Versión simplificada de algunas clases de la biblioteca estandar de SmallEiffel

Para ilustrar el funcionamiento del Análisis de Herencia, consideremos el conjunto de clases de la biblioteca estándar de SmallEiffel que proporcionan la funcionalidad básica de

	is_connected	end_of_input	read_character	last_character	skip_separators	put_character	put_newline	end_of_input	path	connected_to	disconnect	memory
INPUT_STREAM	1	1	1	1	1			1				
OUTPUT_STREAM	1					1	1					
STD_FILE_READ_WRITE	2	2	2	2	2	2	2	2	1	1	1	
STD_FILE_READ	2	2	2	2	2			2	1	1	1	
STD_FILE_WRITE	2					2	2		1	1	1	
STD_INPUT	3	3	3	3	3			3				1
STD_OUTPUT	3					3	3					1
STD_ERROR	3					3	3					1
STD_INPUT_OUTPUT	3	3	3	3	3	3	3	3				1

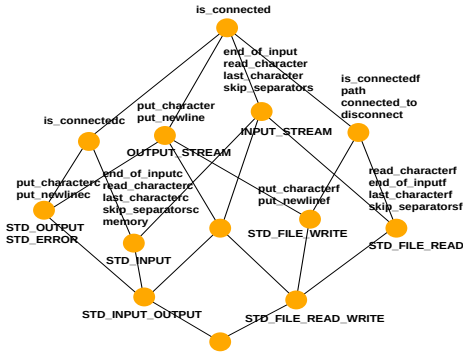

Fig. 3. Tabla de incidencia para el contexto formal extraído de las clases de la Figura 2 y diagrama de Hasse del retículo asociado

entrada y salida de datos. La Figura 2 contiene una representación de este conjunto de clases, simplificado a efectos de facilitar la comprensión de la técnica.

La parte izquierda de la Figura 3 presenta la matriz de incidencia asociada a este conjunto de clases. En esta matriz, las filas representan las clases a analizar, mientras que sus columnas representan los métodos de ésta y la incidencia la presencia de un método en una clase evaluada de acuerdo a la versión del método teniendo en cuenta sus redefiniciones.

La parte derecha de la misma figura refleja el diagrama de Hasse del retículo de Galois obtenido a partir de la matriz de incidencia. Con un análisis adecuado, dicho diagrama permite poner de manifiesto hechos como los siguientes:

- La aparición de un nodo que representa una clase, por debajo del que representa a otra nos indica que todas las características de la primera están presentes en la segunda, por lo que podemos representar a la primera clase como heredera de la segunda. Así en el ejemplo podemos representar STD_FILE_READ_WRITE como heredera de STD_FILE_READ y STD_FILE_WRITE.
- Existen nodos que representan varias clases del conjunto original, y con ello ponen de manifiesto que, con la información recogida en el contexto, las clases son equivalentes. En nuestro ejemplo esto es lo que ocurre con STD_OUTPUT y STD_ERROR.
- Algunos de los nodos no corresponden a clase alguna del conjunto original, pero su presencia proporciona un punto de entrada único para una característica presente en la jerarquía de herencia. En nuestro ejemplo se sugieren varias clases de este tipo, como la que permitiría una definición abstracta para el método `is_connected`.
- Otros nodos que no corresponden a clase o característica alguna del conjunto original pueden constituir una abstracción útil, en el sentido de que recogen los ancestros comunes a varias clases. Así en nuestro ejemplo, se sugiere la oportunidad de introducir una clase que herede de INPUT_STREAM y OUTPUT_STREAM y sirva de ancestro para clases como STD_INPUT_OUTPUT y STD_FILE_READ_WRITE.

El análisis de la relación de cliente permite refinar los resultados de la técnica anterior estructurando la información sobre la forma en que las distintas entidades presentes en el código utilizan las características de su clase base.

Así, a partir del código asociado a las clases representadas en el diagrama de la Figura 4, que recoge una hipotética forma de uso de una versión simplificada de la clase SYSTEM_TIME de la biblioteca estándar de Visual Eiffel, podemos obtener la matriz de incidencia recogida en la parte izquierda de la Figura 5. El ACF permite estructurar esta información en el diagrama que aparece en la parte derecha de la misma figura, del que, con el análisis adecuado, podemos extraer conclusiones como las siguientes.

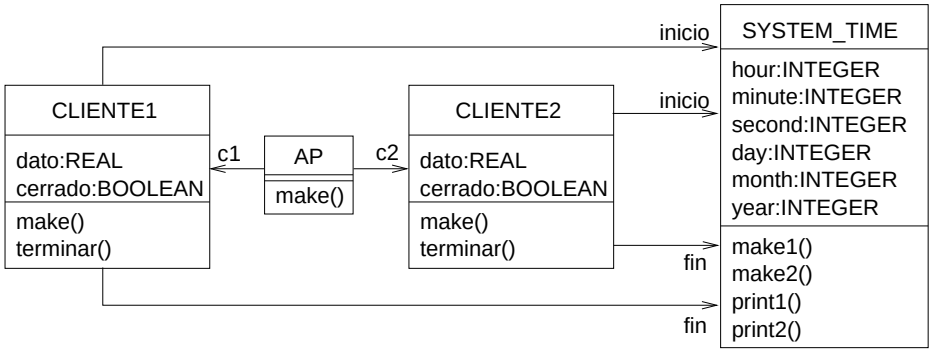

Fig. 4. Clases objeto del análisis de la relación de cliente.

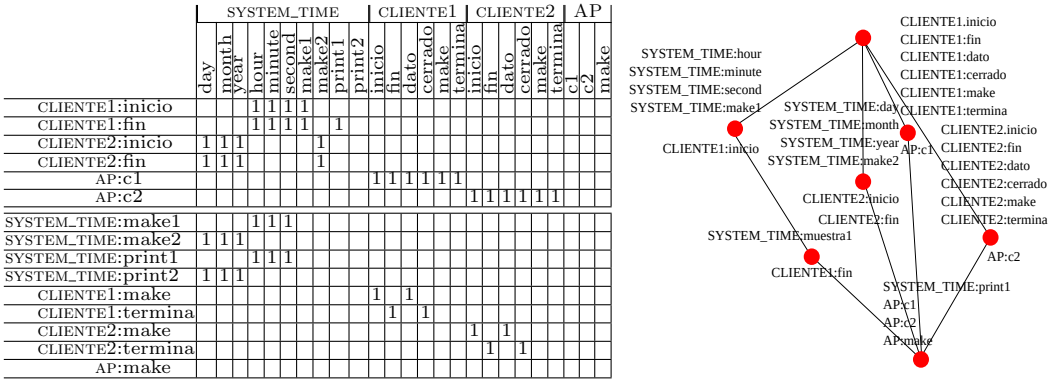

Fig. 5. Contexto asociado a las clases recogidas en la Figura 4 y retículo asociado.

- La aparición de una característica por encima de una entidad, indica que la característica está siendo utilizada a través de la entidad.
- La aparición de entidades con la misma clase base en ramas distintas del retículo nos indica la posibilidad de fraccionar dicha clase. Un claro ejemplo de esto es la clase SYSTEM_TIME.
- Las características que etiquetan el concepto más bajo de este retículo, como `print2`, no están siendo utilizadas por ninguna entidad. Las características de la clase AP son un caso especial, ya que ninguna clase es cliente de ella.

El tercer tipo de técnica basada en el ACF que utilizaremos se denomina Análisis de Dependencias Funcionales, y nos permitirá detectar patrones regulares en la forma en que se instancian los distintos métodos de un punto caliente de caja blanca. Esta información permitirá detectar en el punto caliente las diversas dimensiones de variabilidad que lo componen, y ello facilitará su implementación con técnicas de caja negra.

Para ilustrar la forma en que pretendemos utilizar esta técnica imaginemos un punto caliente de caja blanca en que debemos concretar seis métodos distintos, aunque con ciertas relaciones entre ellos. La forma en que se concretan los métodos en distintas clases puede ser estructurada por las técnicas basadas en ACF. Así ante una instanciación como la recogida en la parte izquierda de la Figura 6 podemos obtener un diagrama como el recogido en la parte derecha de la misma figura, del que con una interpretación adecuada podemos extraer conclusiones como las siguientes:

- Los dos protocolos se determinan mutuamente.
- `interfaz_profesor` determina la versión de los protocolos y de `consulta`.

Clase	prot. acc. interno	definicion	consulta	prot. acc. externo	int. rap. admin.	int. profesor
c1	4	4	4	2	8	1
c2	3	2	2	1	5	2
c3	3	2	3	1	5	3
c4	1	3	4	3	4	4
c5	2	3	3	4	5	5
c6	2	3	3	4	5	5
c7	4	3	2	2	7	6
c8	1	1	2	3	2	7
c9	4	4	1	2	8	8
c10	2	2	2	4	4	9
c11	1	2	1	3	3	0
c12	1	4	2	3	5	7

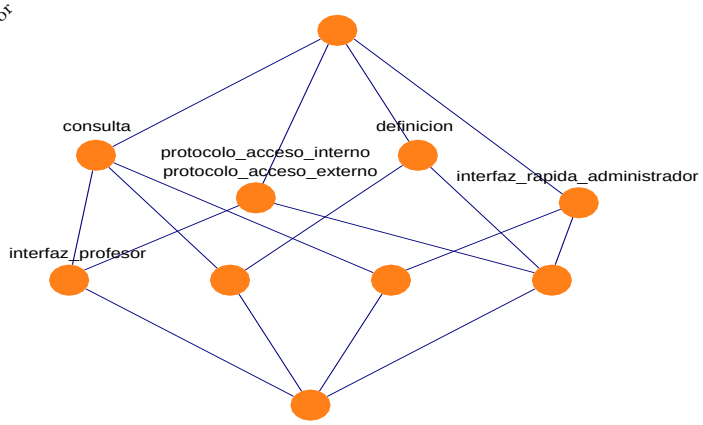

Fig. 6. Contexto de las instanciaciones de un conjunto de métodos y retículo asociado según el Análisis de Dependencias Funcionales.

– Entre `interfaz_profesor` y `definicion` determinan la versión del resto de los métodos.

Podemos concluir que, con la información contenida en las instancias del punto caliente, éste está formado por dos dimensiones de variabilidad, determinadas por `interfaz_profesor` y `definicion`. En cuanto al resto de los métodos, a la luz de la información contenida en las instancias del punto caliente, deberíamos ser capaces de trasladarlas al núcleo del framework e implementarlas en función de la definición de estos dos métodos.

4 Una herramienta experimental basada en ACF

Para que un proceso como el descrito en el apartado 2.1, basado en las técnicas presentadas en la sección anterior, permita la construcción efectiva de frameworks de dominio es imprescindible que se encuentre soportado por herramientas que implementen de forma efectiva las técnicas formales descritas.

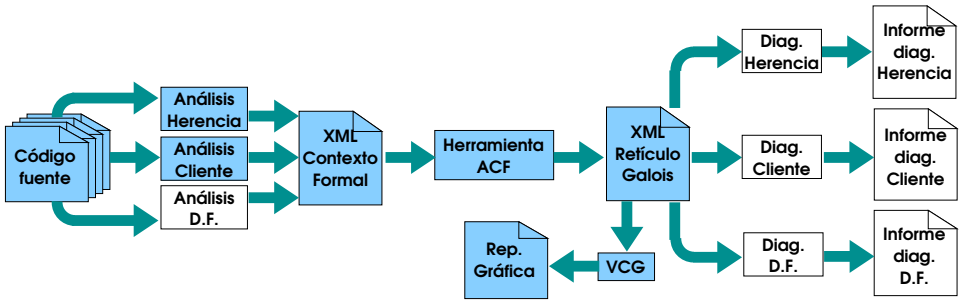

Fig. 7. Arquitectura de la aplicación propuesta.

Estas herramientas deben ser capaces de extraer de forma automática la información presente, tanto en las aplicaciones del dominio como en las instancias del framework, y de presentarla de una forma suficientemente expresiva al desarrollador, ocultándole los detalles técnicos del mecanismo formal que las soporta. En este apartado se presenta la descripción de una herramienta experimental como soporte al proceso descrito, y su utilización para analizar la jerarquía de clases de una biblioteca real, como breve caso de estudio.

El elemento central de estas herramientas estará constituido por la implementación de uno de los algoritmos que proporcionan el retículo asociado a un contexto formal. Existe una gran variedad de algoritmos en estas condiciones, tanto de tipo general, como adaptados a un tipo particular de contexto formal, y se han realizado estudios comparativos entre ellos, como por ejemplo [7]. Para nuestra herramienta experimental, nos hemos decantado por un algoritmo general, el algoritmo de Bordat [2], que auna la facilidad de implementación con un rendimiento razonable. No obstante, la implementación está preparada para la substitución de este algoritmo por otros más efectivos o más adaptados a datos con una configuración característica.

Los datos que alimentan a este algoritmo deben ser extraídos previamente del código fuente objeto del estudio, aplicando los tres modelos de análisis anteriormente expuestos. Esto requiere de una herramienta básica de análisis del código fuente dotada de algoritmos posteriores que identifiquen los datos relevantes para el tipo específico de análisis requerido.

Con el objetivo de garantizar una construcción modular de la herramienta, que facilite la sustitución de algunos módulos, y la reutilización de otros en diferentes procesos, se ha optado por establecer un diseño basado en una arquitectura tipo tubería (pipeline) en la que es importante establecer el formato de la información que se intercambia entre los procesos. La decisión de diseño en este caso es establecer el intercambio mediante documentos XML. El formato de la información de estos documentos se define mediante una DTD propia que permite representar contextos formales y retículos de Galois.

Es posible que en un futuro próximo se defina un estándar XML para este tipo de información que maneja nuestra herramienta[3]. La adopción de este estándar ampliará la facilidad de integración de nuestra herramienta basada en ACF con las desarrolladas por otros autores.

La forma en que ha sido diseñada la herramienta permitirá que un cambio en la sintaxis de los documentos XML resulte simple.

La necesidad de desacoplar tanto como sea posible estas herramientas de extracción de datos y las posteriores de representación del algoritmo de ACF utilizado recomendaba la definición de sendas DTD de XML para representar contextos formales y retículos de Galois. La utilización de XML permite una arquitectura flexible, si bien la definición de una sintaxis propia impide el intercambio de información con herramientas de otros autores. Es por ello que vemos este formato como un elemento provisional, que será modificado si, como parece, se adopta un estándar XML para la representación de estos datos.

La representación de los resultados que proporciona el retículo de Galois debe ser abordada de dos formas distintas.

Las fases iniciales de experimentación, en las que nos encontramos, requieren la representación del retículo de un modo gráfico para su análisis. Para ello hemos optado por utilizar VCG, una herramienta abierta de representación de grafos.

Sin embargo, este análisis requiere de conocimientos sobre la teoría matemática. Para una implantación efectiva en el desarrollo real de software del proceso que proponemos, deben acercarse al desarrollador las conclusiones que se pueden extraer de los retículos. Esto nos obliga a pensar en módulos específicos de diagnóstico capaces de elaborar informes en los términos que son naturales para un desarrollador de software.

De toda esta arquitectura que proponemos, disponemos de implementación para los elementos coloreados. Se dispone de herramientas de análisis de herencia y cliente a partir de código fuente Eiffel, y se encuentran en fase final las mismas herramientas, pero partiendo de código Java y C++.

Los elementos no coloreados se encuentran aún en fases tempranas de desarrollo. La implementación de alguno de los procesos que han permitido construir la herramienta experimental en su versión actual se basa en las implementaciones realizadas en [1,6,14].

Para ilustrar el potencial de todas estas herramientas, podemos mostrar la forma en que se realiza el análisis de la versión real de uno de los ejemplos presentados en este trabajo, el

[3] Al menos si hacemos caso a los comentarios vertidos en la lista de correo `fca-list@indiana.edu`

relativo al análisis de Herencia recogido en las Figuras 2 y 3. Así en el caso del Análisis de Herencia, el conjunto real de nueve clases contiene la definición de 61 métodos, frente a los 12 que contenía la versión reducida, mientras que la matriz de incidencia pasa de 59 a 277 entradas.

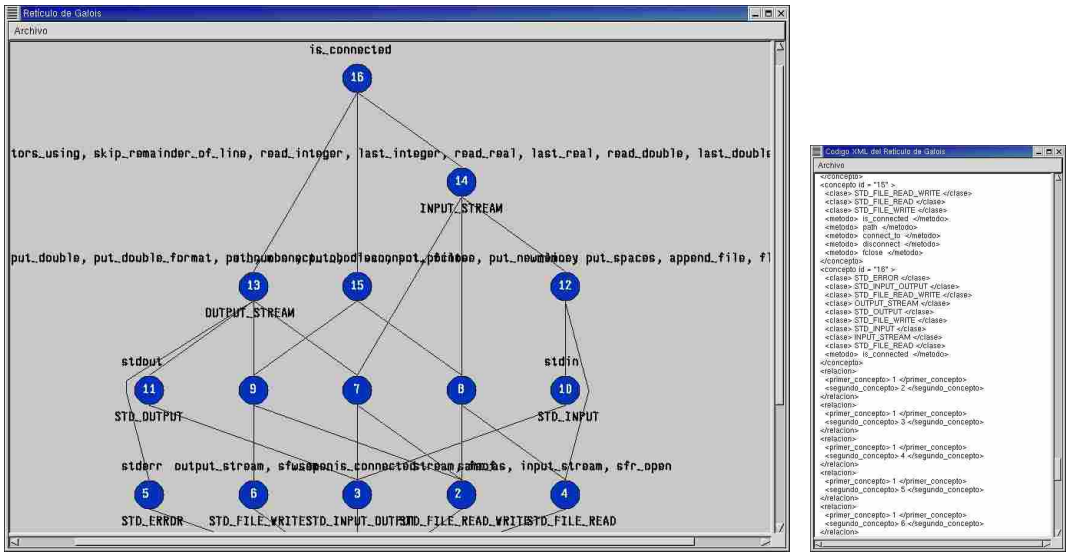

Fig. 8. Resultados de la aplicación del Análisis de Herencia, representados gráficamente y en XML mediante la aplicación experimental realizada

El ACF proporciona un retículo como el recogido en la Figura 8, con un total de 16 conceptos y 27 relaciones entre ellos, lo que sugiere la inserción de 6 nuevas clases en la jerarquía[4], y el desplazamiento de la definición de 32 características.

5 Conclusiones y trabajo futuro

En el presente documento hemos propuesto una línea de trabajo en un proceso de construcción evolutiva aplicable a frameworks de dominio. El enfoque adoptado permite la reutilización del conocimiento almacenado en las aplicaciones del dominio de una manera formal, a diferencia de otras propuestas que simplemente utilizan estas aplicaciones como punto de partida, o como un modo informal de adquirir conocimiento sobre el dominio.

La utilización de técnicas basadas en el ACF permite automatizar el proceso de construcción de un framework, desde su versión inicial, y utilizando la información proporcionada por las aplicaciones del dominio que van siendo implementadas o instanciadas a partir del framework, lo que nos lleva a un enfoque más industrial y por ello más verificable y menos propenso a errores.

En estos momentos disponemos ya del soporte formal tanto para la creación de la versión inicial, como para el proceso de creación de nuevos puntos calientes de caja blanca, y estamos trabajando en la construcción de las herramientas que permitirán la experimentación con un dominio concreto, a partir del código de aplicaciones reales cedido por una empresa.

La continuidad de este trabajo pasa por completar la definición teórica del soporte para la evolución de los puntos calientes desde su forma inicial de caja blanca hasta su versión final

[4] El concepto que se sitúa en el punto más bajo del retículo no puede ser interpretado como una nueva clase en la jerarquía.

de caja negra, y la elaboración de las correspondientes herramientas para la experimentación. Con todo ello podríamos extender la aplicación a nuevos dominios, para intentar confirmar de un modo práctico la viabilidad de este enfoque para el desarrollo de frameworks de dominio.

Referencias

1. J.J Barrio Vizán. Desarrollo de una herramienta de transformación de clases a partir de una operación de parametrización. Proyecto de fin de carrera. dirigido por: Crespo, y. y cardeñoso, v., Universidad de Valladolid, Julio 2000.
2. J. P. Bordat. Calcul partique du treillis de galois d'une correspondance. *Mathématiques et Sciences Humaines*, (96):31–47, 1986.
3. M. Fayad y D. C. Schmidt. Object-Oriented application frameworks. *Communications of the ACM*, 40(10):32–38, Octubre 1997.
4. Mohamed E. Fayad. Introduction to the computing surveys' electronic symposium on object-oriented application frameworks. *ACM Computing Surveys*, 32(1):1–9, Marzo 2000.
5. B. Ganter y R. Wille. *Formal concept analysis: mathematical foundations*. Springer, 1999.
6. M. Gil y J.M. Rodríguez. Herencia múltiple de clases para java. javachm. Proyecto de fin de carrera. dirigido por: Crespo, y.; rodríguez, j.j.; cardeñoso, v., Universidad de Valladolid, September 1999.
7. R. Godin y T. Chau. Comparaison d'algorithmes de construction de hiérarchies de classes. *Aceptado en L'Object*, 2000.
8. R. Godin y H. Mili. Building and maintaining analysis-level class hierarchies using Galois lattices. *Proceedings of OOPSLA'93*, pág. 349–410, 1993.
9. R. Godin, G. Mineau, R. Missaoui, y H. Mili. Méthodes de clasification conceptuelle basées sur les treillis de galois et applications. *Revue d'Intelligence Artificielle*, 9(2):105–137, 1995.
10. R. E. Johnson y B. Foote. Designing reusable classes. *Journal of Object-Oriented Programming*, 1(2):22–35, 1988.
11. Ralph E. Johnson y Vincent F. Russo. Reusing object-oriented designs. Technical Report UIUC DCS 91-1696, University of Illinois, Mayo 1991.
12. W. Pree. *Design Patterns for Object-Oriented Software Development*. Addison Wesley, Wokingham, 1995.
13. Wolfgang Pree. Hot-spot-driven development. En Mohamed Fayad, Douglas Schmidt, y Ralph Johnson, editores, *Building application frameworks: object-oriented foundations of framework desing*, capítulo 16, pág. 379–394. Wiley Computer Publishing, 1999.
14. D. Pérez y F.J. Cano. Implementación de un sistema de diagnóstico de software basado en análisis de conceptos formales. Proyecto de fin de carrera. dirigido por: Crespo, y. y cardeñoso, v., Universidad de Valladolid, febrero 2002.
15. F. Prieto, Y. Crespo, J.M. Marqués, y M.L. Laguna. Mecanos y análisis de conceptos formales como soporte para la construcción de frameworks. En *Actas de las V Jornadas de Ingeniería de Software y Bases de Datos (JISBD'2000)*, pág. 163–175, Noviembre 2000.
16. Don Roberts y Ralph Johnson. Patterns for evolving frameworks. En R. C. Martin, D. Riehle, y F. Buschmann, editores, *Pattern Languages of Program Design*, volume 3, pág. 471–486. Addison–Wesley, 1997.
17. Hans Albrecht Schmid. Framework design by sistematic generalization. En Mohamed Fayad, Douglas Schmidt, y Ralph Johnson, editores, *Building application frameworks: object-oriented foundations of framework desing*, capítulo 15, pág. 353–378. Wiley Computer Publishing, 1999.
18. M. Siff y T. Reps. Identifying modules via concept analysis. *IEEE Transactions on Software Engineering*, 25, Diciembre 1999.
19. G. Snelting y F. Tip. Reengineering class hierarchies using concept analysis. *ACM SIGSOFT Software Engineering Notes*, 23(6):99–110, Noviembre 1998. Proceedings of the ACM SIGSOFT Sixth Internatioal Symposium on the Foundations of Software Engineering.
20. R. Wille. Restructuring lattice theory: An approach based on hierarchies of concepts. En *Ordered Sets*, pág. 445–470. Reidel, Dordrecht–Boston, 1982.
21. R. J. Wirfs-Brock y R. E. Johnson. Surveying current research in Object-Oriented design. *CACM*, 33(9):105–124, Septiembre 1990.