Construcción de frameworks basada en análisis de conceptos formales y soportada por Mecanos.*

Félix Prieto
Dpto. de Informática
Universidad de Valladolid
felix@infor.uva.es

Yania Crespo
Dpto. de Ciencias de la Computación
Universidad de la Habana
yania@infor.uva.es

Francisco José García
Dpto. de Informática y Automática
Universidad de Salamanca
fgarcia@gugu.usal.es

Miguel Angel Laguna
Dpto. de Informática
Universidad de Valladolid
mlaguna@infor.uva.es

Resumen

La creación y evolución de los frameworks es un proceso difícil. En el presente trabajo mostraremos cómo el análisis de conceptos formales puede orientar estos procesos, detectando generalizaciones y sugiriendo reorganizaciones de clases que pueden ser interesantes desde el punto de vista del desarrollador para reutilización. En este contexto el concepto de mecano nos puede proporcionar el soporte para propagar las modificaciones necesarias desde el nivel abstracción en que se detectan.

Palabras clave: Framework, Mecano, Retículo de Galois, Análisis de conceptos formales.

1 Introducción.

La reutilización del software ha sido uno de los principales objetivos de la Ingeniería del Software desde sus orígenes. El paradigma Orientado a Objetos se presentó en su momento como la herramienta definitiva para la obtención de este objetivo, sin embargo, y a pesar de que proporciona avances considerables en este sentido, parece ya evidente que las clases son un elemento reutilizable (*asset*) de grano demasiado fino para basar en ellas el proceso de reutilización [Wirfs-Brock y Johnson, 1990].

Para resolver este problema se han propuesto elementos de grano más grueso como pueden ser los frameworks [Johnson y Foote, 1988] o los mecanos [García, 2000].

Un framework es un diseño reutilizable de todo o parte de un sistema software descrito por varias jerarquías de herencia de clases, generalmente algunas abstractas, y por las colaboraciones que se establecen entre las instancias de estas clases.

La reutilización se implementa entonces mediante la instanciación del framework en una aplicación concreta. Esta instanciación se realiza rellenando los puntos de variabilidad del diseño, los puntos calientes[1] [Pree, 1995], en los que el desarrollador con reutilización incluye la funcionalidad específica de su sistema.

Los frameworks se clasifican en función de la forma en que se instancian [Johnson y Foote, 1988] en:

Frameworks de caja blanca: Para su instanciación, el desarrollador con reutilización necesita conocer su estructura interna. Al utilizarlos, el mecanismo predominante es la herencia, mediante la que se hacen concretos las propiedades abstractas de las clases del framework.

*Este trabajo ha sido parcialmente financiado por la CICYT(TIC97-0593-C05-05) como parte del proyecto MENHIR.

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

Frameworks de caja negra: Para su instanciación no es preciso conocer la forma en que están construidos. El desarrollador con reutilización personaliza los puntos calientes mediante la instanciación con parámetros actuales y la composición.

Una de las características más llamativas de los frameworks es la denominada "inversión de control"por la cual el control del flujo del sistema descansa sobre el código del framework, no sobre el código que incorpora la funcionalidad específica de cada aplicación.

Los frameworks también se clasifican en función del tipo de problema a que van dirigidos [Taligent, 1994] en:

Frameworks de aplicación: Encapsulan una capa de funcionalidad horizontal que se puede aplicar en la construcción de una gran variedad de programas. Los frameworks que implementan las interfaces gráficas de usuario representan su paradigma.

Frameworks de soporte: Proporcionan servicios básicos a nivel de sistema.

Frameworks de dominio: Aplicables a un dominio de aplicación o línea de producto, implementan una capa de funcionalidad vertical. Estos frameworks deberán ser los más numerosos, y su evolución deberá también ser la más rápida, pues deben adaptarse a las áreas de negocio para las que están diseñados.

Los frameworks son, en general, aceptados como un asset de grano adecuado para fomentar el proceso de reutilización, sin embargo su éxito en la práctica ha sido bastante limitado fuera del ámbito de las interfaces gráficas de usuario (frameworks de aplicación), en el que se originaron.

Los principales obstáculos para implantar un modelo de reutilización basado en frameworks vienen dados porque éstos son difíciles de construir y no es sencillo aprender a utilizarlos, y estas dificultades se agudizan en el caso de los frameworks de dominio.

Las dificultades en la creación de frameworks hacen que, si no se preve utilizarlos mucho, sea difícil amortizar el coste de su producción. Por ello 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).
- Se espera que se requieran nuevas aplicaciones del dominio 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 [Roberts y Johnson, 1996] para la fabricación de frameworks que parten de la fabricación de varias aplicaciones del dominio, a partir de las cuales se inicia la construcción de un primer framework de caja blanca que, con la información proporcionada por las sucesivas instanciaciones, va refinándose y transformándose en un framework de caja negra.

La transición se produce también en la dirección contraria, puesto que la evolución del propio dominio de aplicación hará que debamos añadir funcionalidades al diseño del framework, habitualmente mediante la inclusión de nuevas clases abstractas.

El concepto de mecano puede aportar mucho a este modelo evolutivo del desarrollo de frameworks. Se define mecano como un conjunto de elementos software reutilizables clasificados en diferentes niveles de abstracción y relacionados entre sí, ya sea en el mismo nivel (relaciones intranivel) o en niveles diferentes (relaciones internivel), cumpliéndose la restricción de que exista al menos una relación internivel.

Los mecanos, por su propia definición, dan soporte para la trazabilidad entre niveles de abstracción, y un repositorio basado en mecanos, como el repositorio GIRO[2], puede fomentar el aumento del nivel de abstracción de los assets reutilizados.

[2]Grupo de Investigación en Reutilización y Orientación a Objeto."`http://giro.infor.uva.es/`".

Si representamos las aplicaciones iniciales del dominio como un conjunto de mecanos, la separación entre los elementos de diseño e implementación de las mismas puede simplificar notablemente la tarea de abstraer la estructura inicial del framework a partir de los elementos comunes de estas aplicaciones.

A pesar de ello, y gracias a las relaciones internivel, podremos recuperar los elementos de implementación asociados a las abstracciones que conforman el framework.

Sin embargo, para que un repositorio basado en mecanos facilite de modo efectivo la construcción de frameworks, es imprescindible disponer de herramientas que asistan al desarrollador, tanto en la etapa inicial de creación como en la posterior evolución de los mismos.

En el presente trabajo mostraremos cómo las técnicas basadas en análisis de conceptos formales pueden dar soporte a la creación y evolución de los frameworks, detectando generalizaciones y sugiriendo reorganizaciones de clases, que pueden ser interesantes desde el punto de vista del desarrollador para reutilización, y cómo el concepto de mecano nos puede permitir propagar las modificaciones realizadas entre los distintos niveles de abstracción en que deben ser aplicadas.

En lo que sigue, este documento se organiza del siguiente modo: En la siguiente sección introduciremos las definiciones y resultados del análisis de conceptos formales, además de algunas de sus aplicaciones prácticas. En la sección 3 propondremos un proceso de construcción de frameworks asistido por herramientas basadas en el análisis de conceptos formales. Terminaremos la exposición con algunas conclusiones y lineas de trabajo futuro.

2 Análisis de conceptos formales.

El análisis de conceptos formales fue introducido por Wille en [Wille, 1982] y aparece totalmente desarrollado en [Ganter y Wille, 1999]. 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. Inicialmente el análisis de conceptos formales se utilizó en trabajos relacionados con el análisis simbólico de datos y representación del conocimiento (como en [Godin y Missaoui, 1994]). Posteriormente se ha utilizado en temas relacionados con la ingeniería del software, tanto para el estudio de la herencia [Godin y Mili, 1993], como para el estudio de la forma en que se utiliza una biblioteca de clases [Snelting y Tip, 1996], o fuera del paradigma orientado a objetos, para la identificación de módulos sobre código monolítico [Siff y Reps, 1997]. Comenzaremos por introducir las nociones básicas definidas por Wille.

Definición 1 (Contexto formal) *Llamaremos* ***contexto formal*** *a una terna* (G, M, I) *con* $I \subseteq G \times M$.

Informalmente llamaremos a G conjunto de objetos y a M conjunto de atributos. La relación binaria I nos da la incidencia del conjunto de atributos sobre el conjunto de objetos. Habitualmente denotaremos la pertenencia de un par $(g, m) \in G \times M$ al conjunto I de la forma gIm.

La relación binaria nos permite definir un par de aplicaciones mediante las fórmulas

$$\begin{array}{rcl} \varphi : \mathcal{P}(G) & \longrightarrow & \mathcal{P}(M) \\ A & \longmapsto & A^{\uparrow} = \{m \in M \mid gIm \; \forall g \in A\} \\ \psi : \mathcal{P}(M) & \longrightarrow & \mathcal{P}(G) \\ B & \longmapsto & B^{\downarrow} = \{g \in G \mid gIm \; \forall m \in B\} \end{array}$$

y estas dos aplicaciones, contravariantes entre sí, nos permiten realizar la siguiente definición:

Definición 2 (Concepto formal) *Llamaremos* ***concepto formal*** *a un par* $(A, B) \in \mathcal{P}(G) \times \mathcal{P}(M)$ *que verifica* $A^{\uparrow} = B$ *y* $B^{\downarrow} = A$. *Denotamos el conjunto de los conceptos formales asociado a* (G, M, I) *por* $\mathfrak{G}(G, M, I)$.

El concepto formal intenta reflejar la noción informal de concepto como el conjunto de objetos y atributos que se determinan mutuamente.

Sobre (G, M, I) podemos definir una relación de orden parcial mediante la siguiente fórmula en que $(A, B), (A', B') \in \mathfrak{G}(G, M, I)$:

$$(A, B) \leq (A', B') \Leftrightarrow A \subseteq A' (\Leftrightarrow B \supseteq B')$$

El ***terorema básico de los retículos de conceptos*** nos garantiza que el conjunto $\mathfrak{G}(G, M, I)$ junto con la relación de orden dada por la inclusión en los conjuntos de objetos forma un retículo completo. A éste retículo se le denomina retículo de Galois o retículo de conceptos asociado al contexto (G, M, I) y se le denota por $(\mathfrak{G}(G, M, I), \leq)$.

Para representar de una forma clara la información recogida en el retículo de Galois, habitualmente se utiliza el diagrama de Hasse. En este diagrama cada punto representa un concepto formal y cada arco indica una relación de orden entre dos conceptos con el mayor de ellos colocado por encima del menor, con la restricción de que no exista ningún concepto intermedio.

Cuando cada punto se etiqueta con la descripción del concepto asociado, el diagrama se vuelve difícil de leer. Por ello habitualmente cada objeto se coloca etiquetando el concepto más bajo en que está presente dentro del diagrama, $\gamma(g)$, mientras que los atributos etiquetan el más alto en que aparecen dentro del mismo, $\mu(m)$. Por convención las etiquetas relativas a objetos se escriben por bajo el concepto, mientras que las relativas a atributos se sitúan por encima de él.

Con todo ésto, toda la información contenida en la representación original del retículo, y en el contexto formal original, queda reflejada en esta representación. Formalmente:

$$\begin{aligned} \gamma(g) &= \bigvee\{(A, B) \in \mathfrak{G}(G, M, I); g \in A\} \\ \mu(m) &= \bigwedge\{(A, B) \in \mathfrak{G}(G, M, I); m \in B\} \\ (g, m) \in \mathfrak{G}(G, M, I) &\Leftrightarrow \gamma(g) \leq \mu(m) \end{aligned}$$

El interés práctico del análisis de conceptos formales pasa por la disponibilidad de herramientas que automaticen el cálculo del retículo de Galois de un contexto formal. Existen varias publicaciones sobre algoritmos de obtención del retículo de Galois de un contexto formal. Así en [Chaudron y Maille, 1998] aparece un enfoque basado en lógicas de primer orden y programación lógica, mientras que en [Godin et al., 1995] encontramos uno incremental de formación de conceptos. Este algoritmo se compara en [Godin y Chau, 2000] con el propuesto por Dicky y otros. En [Njiwoua y Nguifo, 1997] se paraleliza un algoritmo previo de Bordat.

También existen algunos programas de uso libre para fines no comerciales distribuidos por el grupo de Darmsdadt, como Conimp, que permite entre otras cosas obtener el retículo de Galois aplicando el algoritmo de Bordat, y Diagram, más especializado en la representación gráfica del retículo de Galois.

También existen algunas bibliotecas especializadas en el análisis de conceptos formales como tkconcept, implementado en Tcl/Tk, o la descrita en [Vogt, 1996], implementada en C++.

Toscana es un programa más moderno, basado en la biblioteca de Vogt, que permite explorar interactivamente bases de datos preparadas mediante el análisis de conceptos formales. Anaconda es un editor interactivo pensado para editar los datos que más tarde procesará Toscana.

2.1 Análisis de la herencia.

El análisis de conceptos formales nos permite estudiar la estructura dada por la herencia en un conjunto de clases [Godin y Mili, 1993].

En este estudio el conjunto de objetos G viene dado por los nombres de las clases, mientras que el conjunto de atributos M está formado por los nombres de sus características. La incidencia gIm nos indica que la característica m está presente en la clase g. Los conceptos ahora están definidos por conjuntos de clases y de características que se determinan entre sí, y el retículo de Galois nos permite detectar las siguientes situaciones interesantes:

- Definición del mismo método en distintos puntos del conjunto de clases.

	author	booktitle	chapter	editor	institution	journal	note	pages	publisher	shool	title	year
ARTICLE	✓					✓					✓	✓
BOOK	✓			✓					✓		✓	✓
BOOKLET											✓	
INBOOK	✓		✓	✓				✓	✓		✓	✓
INCOLLECTION	✓	✓							✓		✓	✓
INPROCEEDINGS	✓	✓									✓	✓
MANUAL											✓	
MASTERTHESIS	✓									✓	✓	✓
MISC												
PHDTHESIS	✓									✓	✓	✓
PROCEEDINGS											✓	✓
TECHREPORT	✓				✓						✓	✓
UNPUBLISHED	✓						✓				✓	

Figura 1: Tabla de incidencia para el contexto formal extraído de los tipos de referencias bibliográficas de BIBTEX

- Posibilidad de definir clases abstractas que encapsulan el comportamiento común de un conjunto de clases.

La corrección de estas situaciones en la jerarquía de herencia nos permitirá eliminar la presencia de redundancia y asegurar que la relación de herencia se utiliza básicamente en el sentido de especialización. Todo ello contribuye a aumentar la calidad de la jerarquía de clases [Johnson y Foote, 1988].

Hay que tratar de un modo especial las redefiniciones de características dentro de la jerarquía. Considerar dos versiones de una característica como características esencialmente distintas nos llevaría a perder una información crucial sobre la jerarquía original. Considerar las dos versiones de la característica como idénticas tampoco es razonable, puesto que con ello perdemos la información relativa al nivel de especialización de las mismas.

La alternativa más razonable, propuesta ya en [Godin y Mili, 1993], pasa por elaborar un contexto multivaluado para transformarlo luego en un contexto univaluado, contexto formal o simplemente contexto con la notación de este trabajo, mediante la introducción de una escala que viene dada por la relación de orden parcial que existe entre las distintas versiones del método.

Para ilustrar el potencial de estas técnicas consideremos un imaginario conjunto de clases formado por los tipos de referencia bibliográfica utilizados en BIBTEX, cuyos atributos vienen dados por los campos de información obligatorios en cada tipo de referencia. Podemos construir una matriz de incidencia como la recogida en la figura 1.

El retículo de Galois obtenido aplicando las técnicas de análisis de conceptos formales a la tabla de incidencia de la figura 1 es el recogido en la figura 2, y sobre él podemos hacer algunos comentarios:

- La aparición del nodo que representa una clase por debajo del que representa a otra, $\gamma(g_1) \leq \gamma(g_2)$, nos indica que todas las características de g_2 están presentes en g_1, por lo que podemos considerar a la primera clase heredera de la segunda.

- Algunos de los nodos corresponden a clases del conjunto original y al mismo tiempo a características contenidas en la jerarquía.

- 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.

- Algunos de los nodos no corresponden a clase alguna del conjunto original, pero su presencia proporciona un punto de entrada único para alguna característica presente en la jerarquía de herencia.

- 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.

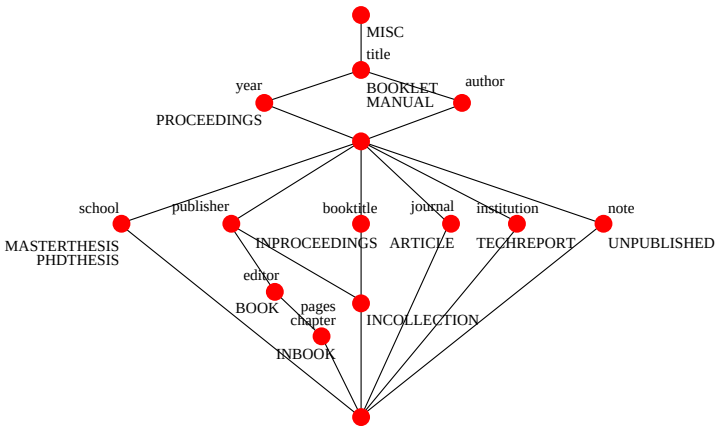

Figura 2: Retículo de Galois de los tipos de referencias bibliográficas de BIBTEX

2.2 Análisis de las colaboraciones entre clases.

El análisis de conceptos formales nos permite analizar y estructurar, además de la herencia, la información relativa a las relaciones de cliente que se establecen entre un conjunto de clases tal y como aparece en [Snelting y Tip, 1996]. Para este tipo de análisis la elaboración de la tabla de datos inicial requiere más información y es más compleja que la anterior.

Hay que resaltar que lo que queremos analizar en este caso no son tan solo las colaboraciones entre clases, sino también qué características en concreto de las clases proveedoras son utilizadas por las clases clientes y en qué modo.

De manera informal podemos decir que en este caso el conjunto de objetos G viene dado por las entidades[3] presentes en las clases a analizar . No se incluyen las entidades cuyo tipo no viene dado por ninguna de las clases que van a ser analizadas.

El conjunto de los atributos M está formado por las características de las clases. Ahora la incidencia $g I m$ pretende reflejar que la entidad g necesita disponer de la característica m.

Para realizar este análisis, a diferencia de lo que ocurría en el anterior, necesitamos la información contenida en el código fuente de las clases, pues de otro modo no podríamos detectar la presencia de relaciones de cliente provocadas por elementos de implementación como las variables locales a los métodos. Precisamente por ello este análisis debe variar ligeramente en función del lenguaje de programación utilizado en cada caso. Nuestra intención en este trabajo es fijar las pautas generales necesarias para un lenguaje orientado a objetos[4], para el que supondremos que dispone de una entidad predefinida, denotada por `Current`, que expresa la autoreferencia de los objetos. Dejaremos los detalles finos para la fase de implementación de los prototipos, que deberán estar orientados a un lenguaje de programación concreto.

En cualquier caso, determinar que la característica m necesita ser definida en la clase base de la entidad g no es una tarea fácil. Si en el código nos encontramos con una llamada como `g.m` la relación es evidente, sin embargo esa no es la única situación que debe ser reflejada en la tabla. Los mensajes no cualificados que aparecen en un método pueden ser considerados como cualificados por la entidad que recibió el mensaje que activó el método, de modo que debemos determinar qué métodos pueden ser invocados desde qué entidades y las incidencias determinadas por ello en la tabla.

Para conseguir este objetivo podemos añadir a la matriz de incidencia una fila extra por cada uno de los métodos definidos en las clases. Estas filas artificiales representan características utilizadas a través del `Current` en dicho método.

Si añadimos reglas que obliguen a reflejar la incidencia $g I m$ también cuando el método m sea

[3] Una entidad es un nombre en el texto de una clase que denota un objeto en tiempo de ejecución, esto es, un atributo, una variable local a un método, un argumento formal de un método o el resultado de una función.

[4] Aunque la terminología utilizada conduce a la notación de Eiffel.

invocado por una entidad g' que, a su vez, puede ser referenciada por g, conseguiremos reflejar la utilización de servicios a través de la entidad g que, de otro modo, quedaba oculta. Para obtener la información de que una entidad g' puede ser referenciada por g necesitaremos hacer un análisis de las ligaduras que ocurren en el texto de las clases.

De un modo en cierto sentido dual, también debemos considerar implicaciones que garanticen la coherencia de los resultados desde el punto de vista de las características de las clases. Así si m' es producto de la redefinición de m debemos considerar que las entidades que deben conocer m' también deben conocer m, pues en otro caso podríamos obtener resultados que nos inclinasen a considerar superfluo m cuando en realidad puede tratarse de un método importante para la definición incremental de m'.

Una vez que hemos construido completamente la matriz de incidencia, las filas correspondientes a las entidades `Current` pueden ser descartadas, puesto que su información estará ya contenida en las filas relativa a otras entidades de las clases, y su presencia en el retículo de Galois final sólo puede contribuir a complicar la interpretación del mismo.

Con todas estas consideraciones el retículo de Galois que podemos construir a partir de la tabla de incidencia descrita nos permitirá obtener interesantes conclusiones sobre las relaciones de cliente que se establecen en el conjunto de clases. Entre ellas cabe destacar:

- Características que no están siendo utilizadas por las entidades instancia de la clase en que están definidas, y que, por ello, pueden ser movidas hacia abajo en la jerarquía, o directamente eliminadas de la misma.
- Clases cuya funcionalidad puede ser repartida entre un conjunto de clases más pequeñas, puesto que las entidades que solicitan sus servicios lo hacen sistemáticamente a un subconjunto de ellos.

La detección de este tipo de situaciones supone la determinación de las refactorizaciones que deben guiar la evolución del conjunto de clases para aumentar su potencial de reutilización, tal y como se indica en [Johnson y Foote, 1988] o [Johnson y Opdyke, 1993].

En las figuras 3, 4 y 5 se recoge un ejemplo de la utilización de esta técnica que ilustra su capacidad para estructurar la información sobre la relación de cliente. Sobre él podemos resaltar:

- La aparición de una característica por encima de una entidad $\mu(m) \geq \gamma(g)$ indica que la característica m está siendo utilizada a través de la entidad g.
- 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 DIA.
- Las características que etiquetan el concepto más bajo de este retículo, como `muestra2`, no están siendo utilizadas por ninguna entidad. Las características de la clase AP son un caso especial, puesto que ninguna clase es cliente de ésta.

3 Proceso de construcción de frameworks.

El análisis de conceptos formales puede ser la base para una buena herramienta de asistencia para la construcción de frameworks, a partir de un conjunto de aplicaciones del dominio, o para su posterior evolución. Las técnicas presentadas en la sección anterior pueden ser útiles para:

- Unificar las jerarquías de herencia presentes en las soluciones de problemas del dominio de las que partimos. El estudio del conjunto de clases mediante la técnica basada en herencia permite poner de manifiesto la existencia de clases que comparten el mismo conjunto de características.

```
class DIA
creation make1,make2
feature
  hora,minuto,segundo:INTEGER
  dia,mes,anno:INTEGER
  make1(h,m,s:INTEGER) is
    do
    hora:=h; minuto:=m; segundo:=s
    end —— make1
  make2(d,m,a:INTEGER) is
    do
    dia:=d; mes:=m; anno:=a
    end —— make2
  muestra1 is
    do
      print(hora);print(minuto);print(segundo)
    end
  muestra2 is
    do
      print(dia);print(mes);print(anno)
    end
end —— Class DIA
```

```
class AP
creation make
feature
  c1:CLIENTE1
  c2:CLIENTE2
  make is
    do
     !!c1.make(1,1,1,0)
     !!c2.make(1,1,1,0)
     c1.terminar(1,1,1)
     c2.terminar(1,1,1)
    end
end —— Class AP
```

```
class CLIENTE1
creation make
feature
  inicio,fin:DIA
  dato:REAL
  cerrado:BOOLEAN
  make (h,m,s:INTEGER; r:REAL) is
  do
    !!inicio.make1(h,m,s)
    dato:=r
  end
  terminar(h,m,s:INTEGER) is
  do
    !!fin.make2(h,m,s); fin.muestra1
    cerrado:=False
  end
end—— Class CLIENTE1
```

```
class CLIENTE2
creation make
feature
  inicio,fin:DIA
  dato:REAL
  cerrado:BOOLEAN
  make (d,m,a:INTEGER; r:REAL) is
  do
    !!inicio.make2(d,m,a)
    dato:=r
  end
  terminar(d,m,a:INTEGER) is
  do
    !!fin.make2(d,m,a)
    cerrado:=False
  end
end—— Class CLIENTE2
```

Figura 3: Código de las clases objeto del análisis

	DIA										CLIENTE1						CLIENTE2						AP		
	dia	mes	anno	hora	minuto	segundo	make1	make2	muestra1	muestra2	inicio	fin	dato	cerrado	make	terminar	inicio	fin	dato	cerrado	make	terminar	c1	c2	make
CLIENTE1:inicio				✓	✓	✓	✓																		
CLIENTE1:fin				✓	✓	✓	✓		✓																
CLIENTE2:inicio	✓	✓	✓					✓																	
CLIENTE2:fin	✓	✓	✓					✓																	
AP:c1											✓	✓	✓	✓	✓	✓									
AP:c2																	✓	✓	✓	✓	✓	✓			
DIA:make1				✓	✓	✓																			
DIA:make2	✓	✓	✓																						
DIA:muestra1				✓	✓	✓																			
DIA:muestra2	✓	✓	✓																						
CLIENTE1:make											✓		✓												
CLIENTE1:terminar												✓		✓											
CLIENTE2:make																	✓		✓						
CLIENTE2:terminar																		✓		✓					
AP:make																									

Figura 4: Contexto asociado a las clases recogidas en la figura 3

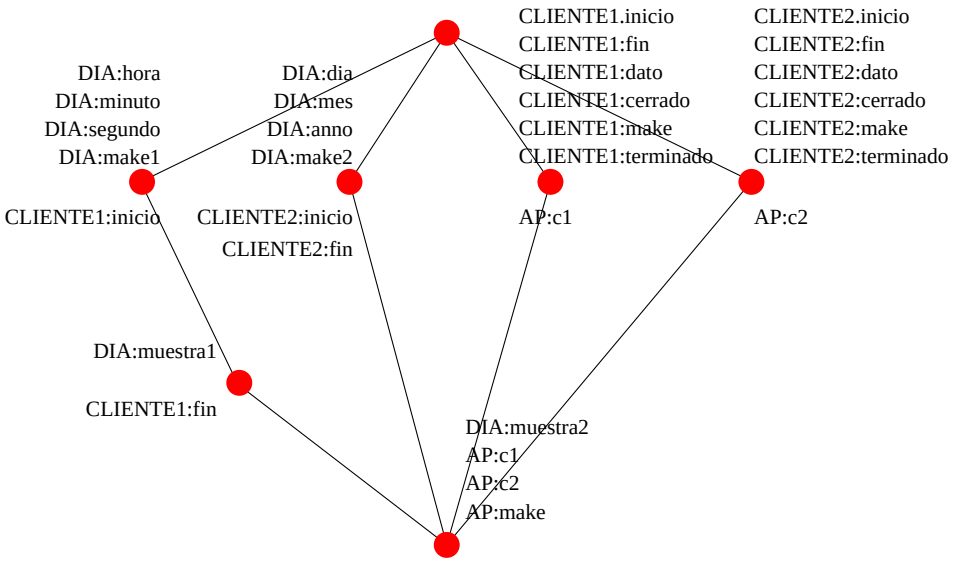

Figura 5: Retículo de Galois asociado al contexto formal de la figura 4

- Cuando no sea posible unificar varias clases, el mismo tipo de análisis nos ayudará a poner de manifiesto las abstracciones comunes a las mismas, sirviéndonos además de índice sobre la idoneidad de las jerarquías de herencia utilizadas.

- Estudiar las relaciones de cliente existentes, sugiriendo al diseñador la posibilidad de eliminar características, mover su definición entre clases, o fraccionar una clase en clases más pequeñas y reutilizables.

La existencia de técnicas aplicables a elementos con distinto nivel de abstracción nos sugiere la oportunidad de basar este proceso en el mecano, como soporte de la trazabilidad entre diferentes niveles de abstracción.

El proceso de construcción de frameworks debería entonces comenzar por el almacenamiento de algunas aplicaciones del dominio en un repositorio como el soportado por el grupo GIRO, estructurándolas como un conjunto de mecanos. El resultado final de todo el proceso sería entonces también un mecano que constituye un framework.

En una primera aproximación, las fases en que se dividiría el proceso son las siguientes:

1. Recopilar el conjunto de clases del dominio del problema que participan en la solución de aplicaciones de la línea de producto a partir de los documentos de diseño disponibles. Ésta recuperación se puede ver facilitada si se inicia a partir del nivel de requisitos dentro del repositorio GIRO.

   Lo que nos interesa en esta primera fase son los elementos de diseño de estas aplicaciones, porque en ellos está contenida toda la información que necesitaremos en los tres siguientes pasos: Los nombres de las clases y las signaturas de sus características.

2. Normalizar las signaturas de las características de las clases. El análisis de conceptos formales nos sugerirá la posibilidad de unificar varias clases cuando compartan las mismas características, pero su único índice sobre la igualdad entre ellas viene dado por la información que le proporcione en este momento el desarrollador para reutilización.

3. Determinar un orden parcial sobre las características que, tras la fase anterior, mantienen el mismo nombre. Este orden parcial debe reflejar la especialización progresiva de las características en la jerarquía de herencia, y será respetado por el algoritmo de análisis de conceptos formales.

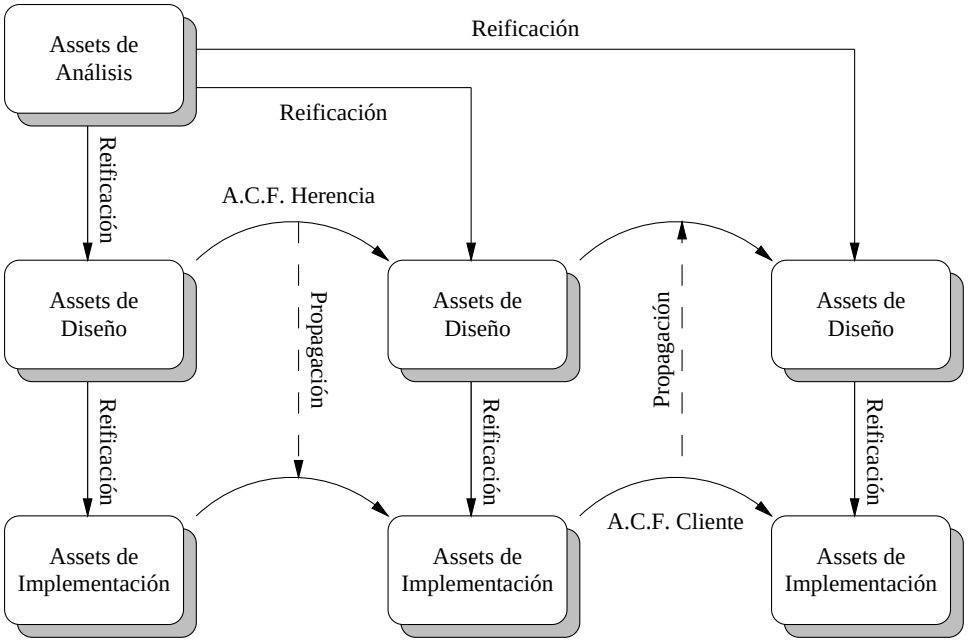

Figura 6: Esquema del proceso propuesto.

4. Aplicar el análisis de conceptos formales para estudiar las jerarquías de herencia a nivel de diseño obtenidas en los pasos anteriores.

5. Una vez aplicadas las refactorizaciones que se consideren interesantes de entre las sugeridas por el paso anterior, propagarlas al nivel de implementación y obtener el código asociado con las clases analizadas mediante las relaciones internivel dadas por la estructura de mecano.

6. Aplicar el análisis de conceptos formales al conjunto de clases de implementación obtenido en el apartado anterior para estudiar las relaciones de cliente que se establecen entre las clases objeto de estudio.

7. Una vez aplicadas las refactorizaciones que se consideren adecuadas de entre las indicadas por el paso anterior, propagarlas a los diseños asociados al código por las relaciones internivel.

Para que este esquema, ilustrado en la figura 6, sea plenamente operativo será imprescindible disponer de una herramienta versátil de obtención del retículo de Galois asociado a un contexto formal, así como de herramientas capaces de obtener el contexto formal a partir de la información contenida en el repositorio GIRO. También será necesaria una herramienta de visualización del retículo de Galois capaz de presentar la información de manera comprensible para el desarrollador.

4 Conclusiones y trabajo futuro.

En el presente trabajo hemos mostrado cómo el concepto de mecano permite relacionar dos técnicas de estudio de clases basadas en el análisis de conceptos formales. Esta posibilidad sugiere que la construcción, y la posterior evolución de los frameworks de dominio, puede ser facilitada por la utilización de herramientas de asistencia al desarrollador para reutilización basadas en estas técnicas.

La continuidad de este trabajo pasa por definir formalmente restricciones para los mecanos que nos permitan representar con garantías un framework. Ello nos permitiría abordar la construcción de prototipos que implementen las funcionalidades descritas en este trabajo, y abordar con ellos la experimentación basada en las aplicaciones sobre tratamiento digital de imagen o gestión universitaria que están ya almacenadas en el repositorio GIRO.

La construcción de estas herramientas requiere los siguientes elementos:

- Elaboración de una herramienta versátil de análisis de conceptos formales, capaz de elaborar el retículo de Galois de un contexto formal.
- Construcción de herramientas capaces de obtener los contextos a partir de la información almacenada en el repositorio GIRO. Estas herramientas deberán estar especializadas en el tipo de información a analizar, de herencia o de cliente, y, en el último caso, en el lenguaje de representación de los assets de modelado e implementación elegido.

  Para este segundo tipo de herramientas podemos basarnos en el trabajo previo sobre el análisis de código Eiffel y Java, que nos proporcionan los prototipos construidos para la aplicación de las refactorizaciones definidas en [Crespo, 2000]
- Además sería necesaria una herramienta de visualización capaz de mostrar de forma clara toda la información contenida en el retículo, resaltando las propuestas de reestructuración que se deducen de éste.
- Por último deberemos fabricar herramientas capaces de aplicar las refactorizaciones sugeridas por el análisis de conceptos formales para lo cual podemos apoyarnos, por ejemplo, en los trabajos de Opdyke[Opdyke, 1992] , Moore [Moore, 1996], Crespo [Crespo, 2000].

Referencias

[Chaudron y Maille, 1998] Chaudron, L. y Maille, N. (1998). 1st order logic formal concept analysis: from logic programming to theory. *Linhöping Electronic Articles in Computer and Information Science*, 3(13).

[Crespo, 2000] Crespo, Y. (2000). *Refactorizaciones motivadas por la reutilización en un modelo mínimal, con variantes, de lenguajes de programación orientados a objetos*. Tesis doctoral, Universidad de Valladolid, ***actualmente en ejecución***.

[Ganter y Wille, 1999] Ganter, B. y Wille, R. (1999). *Formal Concept Analysis: Mathematical Foundations*. Springer.

[García, 2000] García, F. (2000). *Modelo de reutilización soportado por estructuras complejas de reutilización denominadas mecanos*. Tesis doctoral, Dpto. de Informática y Automática. Universidad de Salamanca.

[Godin y Chau, 2000] Godin, R. y Chau, T.-T. (2000). Comparaison d'algorithmes de construction de hiérarchies de classes. *Aceptado en L'Object*.

[Godin y Mili, 1993] Godin, R. y Mili, H. (1993). Building and maintaining analysis-level class hierarchies using galois lattices. *OOPSLA*, pages 349–410.

[Godin y Missaoui, 1994] Godin, R. y Missaoui, R. (1994). An incremental concept formation approach for learning from databases. *Theoretical Computer Science*, 133(2):387–419.

[Godin et al., 1995] Godin, R., Missaoui, R., y Alaoui, H. (1995). Incremental concept formation algorithms based on galois (concept) lattices. *Computational Intelligence*, 11(2):246–262.

[Johnson y Opdyke, 1993] Johnson, R. y Opdyke, W. (1993). Refactoring and aggregation.(invited paper). In Nishio, S. y Yonezawa, A., editors, *Object Technologies for Advanced Software. Proceedings of the 1st JSSST International Symposium*, pages 264–278. Springer-Verlag, LNCS 742.

[Johnson y Foote, 1988] Johnson, R. E. y Foote, B. (1988). Designing reusable classes. *Journal of Object-Oriented Programming*, 1(2):22–35.

[Moore, 1996] Moore, I. (1996). Automatic inheritance hierarchy restructuring and method refactoring. *OOPSLA*.

[Njiwoua y Nguifo, 1997] Njiwoua, P. y Nguifo, E. M. (1997). A parallel algorithm to build concept lattice. In *4th Groningen Intl. Information Tech. Conf. for Students*, pages 103–107.

[Opdyke, 1992] Opdyke, W. (1992). *Refactoring Object-Oriented Frameworks*. Tesis doctoral, Department of Computer Science, University of Illinois at Urbana-Champaign. also Technical Report UIUCDCS-R-92-1759.

[Pree, 1995] Pree, W. (1995). *Design Patterns for Object-Oriented Software Development*. Addison Wesley, Wokingham.

[Roberts y Johnson, 1996] Roberts, D. y Johnson, R. (1996). Evolving frameworks: A pattern language for developing object-oriented frameworks. http://st-www.cs.uiuc.edu/ droberts/evolve.html.

[Siff y Reps, 1997] Siff, M. y Reps, T. (1997). Identifying modules via concept analysis. Technical report cs-tr-97-1337, University of Wisconsin-Madison.

[Snelting y Tip, 1996] Snelting, G. y Tip, F. (1996). Reengineering class hierarchies using concept analysis. *ACM Transactions on Software Engineering and Methodology*, 5(2):146–189.

[Taligent, 1994] Taligent (1994). Building object-oriented frameworks. White paper, Taligent Inc.

[Vogt, 1996] Vogt, F. (1996). *Datenstrukturen und Algorithmen zur Fromalen Begriffsanalyse: Eine C++ Klassenbibliothek*. Springer-Verlag, Berlin-Heidelberg-New York.

[Wille, 1982] Wille, R. (1982). *Restructuring Lattice Theory: An Approach Based on Hierarchies of concepts*, pages 445–470. I.Rival.

[Wirfs-Brock y Johnson, 1990] Wirfs-Brock, R. J. y Johnson, R. E. (1990). Surveying current research in object-oriented design. *CACM*, 33(9):105–124.