# Un estudio de técnicas de estructuración de software: aplicación en el ambiente Adele

N. Belkhatir R. Casallas W. L. Melo
Laboratoire de Génie Informatique
BP 53X
38.041 Grenoble Cedex France
e-mail: casallas@imag.imag.fr
phone: +(33) 76 51 46 68

**Resumen**

Este artículo presenta un estado del arte sobre la utilización de técnicas de clasificación para el mantenimiento de software de gran tamaño. Estas técnicas permiten hacer reagrupamientos sobre un conjunto de objetos atómicos. Para hacer esto, muchas de ellas se apoyan sobre el domino de la taxonomía numérica y sobre técnicas de inteligencia artificial. Nosotros presentamos las caracteristícas principales de las técnicas de clasificación numérica y conceptual. Mostramos cinco ejemplos de herramientas que utilizan este enfoque. Las dos primeras utilizan las técnicas llamadas numéricas: la primera para encontrar las partes de un software suceptible de contener errores, la segunda, para proponer un reordenamiento para la integración del software modificado. Las tres siguientes utilizan las técnicas llamadas conceptuales. Su propósito es, respectivamente, encontrar la estructura lógica del software viejo, optimizar los procesos de reutilización y generar automáticamente bibliotecas de códigos reutilizables. Después de una comparación de estas diferentes técnicas, mostramos como nosotros podemos utilizarlas dentro del ambiente Adele. Terminamos por una evaluación sucinta de experimentos realizados.

**Palabras claves**

Técnicas de clasificacíon, mantenimiento de software de gran tamaño, reutilización, reestructuración, taxonomía numérica, técnicas de clasificación conceptual, programación global, Adele, métricas.

# 1 Introducción

El costo de mantenimiento y evolución del software está en progresión constante. Numerosos estudios [Pressman87] situan, para el próximo decenio, el costo de la actividad de mantenimiento aproximadamente como en el 80% del costo total del software existente. De ahí la necesidad de proponer y de desarrollar nuevos métodos y herramientas para ayudar en la automatización de esta importante etapa del ciclo de vida del software. Este artículo propone enfoques para la reutilización y el mantenimiento de la estructura del software de gran tamaño que se basan en técnicas de clasificación. Estas técnicas permiten obtener una estructura del software que esta en servicio, a partir del código fuente.

El software voluminoso (superior a $10^6$ lineas de código fuente) tienen una gran vida útil en comparación con el tiempo invertido en su desarrollo. Durante este periodo, las modificaciones efectuadas, cualquiera que sea su tipo, (correctivas, evolutivas, adaptativas) desestructuran progresivamente el software. Las herramientas que se basan sobre las técnicas de clasificación permiten a partir del código fuente:

- de una parte, extraer la estructura real del software,
- y de otra, comparar con la estructura inicial del diseño para señalar los cambios eventuales.

Estos cambios necesitan una interpretación, que puede ser automatizada, para detectar las eventuales incoherencias en la evolución.

Otra aplicación de las técnicas de clasificación es la que concierne con los aspectos de reutilización de componentes de software. Paro esto, se utilizan las técnicas de indexación automática para derivar las propiedades de los componentes a partir de su documentación.

El artículo presenta las principales técnicas, ilustradas en ejemplos de herramientas y utilizadas dentro del contexto de mantenimiento de software. La sección 2 introduce las técnicas de clasificación. Mostramos sus principales características y funciones. La sección 3 presenta cinco ejemplos de estas técnicas en la etapa de mantenimiento. La sección 4 presenta una evaluación comparativa de experimentos realizados. La sección 5 describe brevemente el ambiente Adele y la evaluación de los primeros experimentos llevados a cabo, los cuales aplican las técnicas de estructuración al software que Adele administra.

# 2 Las técnicas de clasificación

Las técnicas de clasificación tienen por objetivo organizar conjuntos de datos que representan por ejemplo: medidas, observaciones, eventos. Esto consiste en formar clases donde en cada una se agrupan los objetos que tienen una fuerte similitud. Después [Fischer86], un modelo general puede ser definido como sigue:

**Dados:** un conjunto de objetos O.

**Objetivo:** identificar las subclases $s_1, \ldots, s_i, \ldots, s_n$ (subconjuntos de O), tales que la similitud de los objetos de cada $s_i$ tiende a ser maximizada mientras que la similitud de los objetos de subclases diferentes tiende a ser minimizadá. La colección de clases se llama una clasificación [1].

[1] La organizació de clases y su estructura interna cambian en función de la técnica escogida. Cada aplicación tiene una técnica apropiada

Hemos identificado dos tipos de técnicas de clasificación:

- las técnicas llamadas de taxonomía numérica [Everitt80,skodal63,skodal88]. También se encuentra la expresión clasificación numérica para denotar esta técnica.
- las técnicas llamadas de clasificación conceptual [2] [Michalski80] desarrolladas recientemente y más generales en comparación con las técnicas numéricas.

## 2.1 Taxonomía numérica

Las medidas utilizadas para identificar la similitud entre los objetos es de naturaleza numérica. El conjunto de atributos que caracterizan los objetos es limitado y definido a priori. La similitud entre los objetos es evaluada utilizando una matriz de similitud. Según [Hutchens85], una matriz de similitud D, para un conjunto ordenado P de n elementos, es una matriz cuadrada D[n,n] tal que:

$$D(i,j) \geq 0 \qquad 1 \leq i \leq n \qquad 1 \leq j \leq n$$
$$D(i,j) = D(j,i)$$
$$D(i,i) = 0$$

Según [Fisher86], las técnicas numéricas pueden ser estructuradas en:

- Técnicas de optimización.
  Estas técnicas intentan formar una cantidad óptima de K particiones sobre un conjunto de objetos O. O es dividido en K grupos mutuamente excluyentes. El valor de K es dado por el usuario.
- Técnicas jerárquicas.
  Estas técnicas intentan crear un árbol de clasificación binaria, llamado dendogramme, sobre un conjunto de objetos O. Las hojas representan los objetos atómicos y los nodos internos los grupos de objetos. Las técnicas jerárquicas pueden todavía ser divididas en técnicas llamadas de aglomeración y de división. Estas últimas construidas en forma de dendogramme de forma ascendente o descendente.
- Técnicas de recubrimiento [3].
  Estas técnicas intentan construir grupos de objetos no mutuamente excluyentes.

## 2.2 Técnica Conceptual

Las técnicas de clasificación conceptual se basan sobre los atributos cuantitativos (medidas numéricas) y cualitativos que describen las propiedades. El objetivo de una clasificación conceptual no es sólo identificar las clases, sino también, determinar la estructura conceptual implícita. Se define una clase por la enumeración de sus elementos y por un conjunto de propiedades que definen las relaciones entre sus miembros.

Un estudio mas detallado de técnicas de clasificación conceptual y de ejemplos de comparación con la taxonomía numérica son presentados en [Fisher86, Michalski83]. [Fisher86] compara varios sistemas de clasificación conceptual y [Michalski83] describe con mas detalle el sistema que se basa sobre la técnica Cluster/2.

[2] "Conceptual clustering"
[3] "Cluping"

# 3 Ejemplos de aplicación en el dominio de la ingenieria de software

En esta parte presentamos cinco ejemplos de herramientas que utilizan las técnicas de clasificación para ayudar en el mantenimiento de software de gran tamaño. Los dos primeros ejemplos utilizan la técnica de clasificación numérica para: (1) identificar las partes del software sujetas a error, (2) aportar una ayuda en la integración de software de gran tamaño. El tercer ejemplo utiliza una técnica de clasificación conceptual para generar nuevas estructuras jerárquicas del sistema o para hacer la comparación entre la estructura producida y la deseada. El cuarto y quinto ejemplo utilizan técnicas conceptuales diferentes para crear y mantener bibliotecas de código reutilizable.

Hemos escogido ejemplos de trabajos recientes y aplicables en la fase de mantenimiento. Los tres primeros ejemplos utilizan como único recurso el código fuente del software. Los dos últimos utilizan otros tipos de información como por ejemplo los índices derivados de la documentación por una herramienta de indexación automática.

Este trabajo describe experiencias de la aplicación de estas técnicas en el dominio de la ingenieria de software. No discutimos los algoritmos utilizados ya que estos estan suficientemente documentados (cf. bibliografía).

## 3.1 Identificación de las partes del software que contienen errores

[Selby88a y 88b] utiliza un técnica de clasificaión numérica para extraer del código fuente una estructura jerárquica del sistema. Los autores utilizán, como criterio de clasificación, medidas de interacción para un software distribuido en diversos tipos. La tabla 1 muestra algunos ejemplos de tipos de interacción definidos como sigue:

- Interacción potencial: está definida como una tupla (p,x,q) donde p y q son son procedimientos y x es una variable global para p y q.
- Interacción por referencia: es una interacción potencial donde los procedimientos p y q utilizan la variable x.
- Interacción por afectación: es una interacción por referencia donde el procedimiento p cambia el valor de x y el procedimiento q lo consulta. El orden de ejecución de p y q no es tenido en cuenta.
- Interacción por flujo de datos: es una interacción por afectación que permite llamar al procedimiento p a partir del procedimiento q. Este llamado es controlado por una condición sobre la variable x.

Los autores utilizan un conjunto integrado de cinco herramientas para convertir el código fuente en una descripción jerárquica del sistema [Selby88b, Hutchens87]. La primera herramienta es la encargada de extraer de cada módulo la relación de utilización entre las variables y los procedimientos. La segunda crea, a partir de los resultados de la primera herramienta, una relación de utilización única para todo el software. La tercera herramienta genera una matriz, donde cada procedimiento tiene asociado una fila y una columna, y donde el contenido de la matriz es la similitud, i.e. la cantidad de interacciones por afectación entre

| Código Fuente | Interacción potencial | Interacción por referencia | Interacción por afectación | Interacción por flujo de datos |
|---|---|---|---|---|
| INT a,b,c,d | (p4,a,p1), (p4,a,p2), (p4,a,p3) | (p1,a,p2) | (p1,a,p2) | (p1,a,p2) |
| PROC p1 | (p1,b,p2), (p1,b,p3), (p1,b,p4) | (p2,a,p1) | (p2,b,p1) | (p2,b,p1) |
| /* uses a,b */ | (p2,b,p1), (p2,b,p3), (p2,b,p4) | (p1,b,p2) | (p3,c,p4) | (p3,c,p4) |
| /* assigns a */ | (p3,b,p1), (p3,b,p2), (p3,b,p4) | (p2,b,p1) | (p4,d,p3) | (p2,e,p3) |
| . . . . . | (p4,b,p1), (p4,b,p2), (p4,b,p3) | (p3,c,p4) | (p2,e,p3) | |
| CALL p2 | (p1,c,p2), (p1,c,p3), (p1,c,p4) | (p4,c,p3) | | |
| . . . . . | (p2,c,p1), (p2,c,p3), (p2,c,p4) | (p3,d,p4) | | |
| PROC p2 | (p3,c,p1), (p3,c,p2), (p3,c,p4) | (p4,d,p3) | | |
| /* uses a,b */ | (p4,c,p1), (p4,c,p2), (p4,c,p3) | (p2,e,p3) | | |
| /* assigns b */ | (p1,d,p2), (p1,d,p3), (p1,d,p4) | | | |
| . . . . . | (p2,d,p1), (p2,d,p3), (p2,d,p4) | | | |
| CALL p3(x) | (p3,d,p1), (p3,d,p2), (p3,d,p4) | | | |
| . . . . . | (p4,d,p1), (p4,d,p2), (p4,d,p3) | | | |
| CALL p4 | (p1,e,p3), (p2,e,p3), (p4,e,p3) | | | |
| . . . . . | | | | |
| PROC p3 (int e) | | | | |
| /* uses c,d,e */ | | | | |
| /* assigns c */ | | | | |
| . . . . . | | | | |
| PROC p4 | | | | |
| /* uses c,d */ | | | | |
| /* assigns d */ | | | | |
| . . . . . | | | | |
| START p1 | | | | |

Tabla 1 - Ejemplo de interacción de datos [Hutchens85].

Matríz de interacción de datos por afectación (denotado Inter)

| | p1 | p2 | p3 | p4 |
|---|---|---|---|---|
| p1 | 0 | 2 | 0 | 0 |
| p2 | 2 | 0 | 1 | 0 |
| p3 | 0 | 1 | 0 | 2 |
| p4 | 0 | 0 | 2 | 0 |

Matriz de similitud (denotado Simil)

| | p1 | p2 | p3 | p4 | |
|---|---|---|---|---|---|
| p1 | 0 | 1/2 | I | I | I = ∞ |
| p2 | 1/2 | 0 | 5/3 | I | |
| p3 | I | 5/3 | 0 | 1/2 | |
| p4 | I | I | 1/2 | 0 | |

Utilizando el ejemplo mostrado en la table 1, la matríz de similitud se construyó con los elementos definidos como sigue:

Simil(i,j) = (k/(n-1)) / Inter(i,j), où: n = cantidad de procedimientos

$k = \sum \text{Inter}(i, m) + \sum \text{Inter}(m, j) - \text{Inter}(i, j)$,

m=1,..., n.

A partir de estas matrices, la técnica utlizada construye una jerarquía como sigue:

1 (p1 p 2 p3 p4)
1/2 (p1, p2)
1/2 (p3, p4)

Los grupos (p1, p2) y (p3, p4) sont logrados en la primera iteración del algoritmo. Después en cada iteración la matríz de similitud es actualizada y se forman de nuevo los grupos. Esta técnica ha sido aplicado para descomponer grandes sistemas en subsistemas. [Belady82].

Tabla 2 -Ejemplo de clasificación para las medidas de flujos de datos. [Hutchens85].

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CMDDATA | 0 | | | | | | | | | | | | |
| CMDS | 27 | 0 | | | | | | | | | | | |
| COMPILE | 17 | 8 | 0 | | | | | | | | | | |
| CONCUR. | 2 | 13 | 3 | 0 | | | | | | | | | |
| DBLOGIC | 31 | 23 | 31 | 4 | 0 | | | | | | | | |
| DBMNGT | 42 | 51 | 10 | 1 | 25 | 0 | | | | | | | |
| RESERV. | 18 | 9 | 0 | 0 | 1 | 6 | 0 | | | | | | |
| UPDATE | 5 | 21 | 52 | 3 | 30 | 46 | 2 | 0 | | | | | |
| UTILITY | 27 | 21 | 28 | 2 | 0 | 6 | 0 | 31 | 0 | | | | |
| MSG | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | |
| PROFILE | 0 | 2 | 1 | 3 | 0 | 2 | 0 | 0 | 3 | 0 | 0 | | |
| SYNCH | 2 | 5 | 0 | 0 | 0 | 4 | 1 | 0 | 0 | 0 | 0 | 0 | |
| UTIL | 8 | 17 | 9 | 0 | 5 | 19 | 4 | 4 | 14 | 3 | 1 | 6 | 0 |
| | CMDATA | CMDS | COMPILE | CONCUR | DBLOGIC | DBMNGT | RESERV. | UPDATE | UTILITY | MSG | PROFILE | SYNCH | UTIL |

Figura 1 - Matríz de similitud del sitema Smile [Maarek88a].

una pareja de procedimientos. La cuarta heramienta construye la matriz de similitud ( una matriz no negativa, real, simétrica con ceros en la diagonal). La quinta herramienta utiliza una técnica estática de clasificación jerárquica [Everitt80], produce una descripción arborescente, donde los procedimientos son colocados como hojas del árbol y cada subárbol indica una clase de procedimientos que tienen en común un gran nivel de acoplamiento/cohesión. El acoplamiento es la cantidad de conexiones entre los elementos de un grupo con los otros grupos. La tabla 2 muestra un ejemplo de utilización.

## 3.2 Integración de Software

[Maarek88a y 88b] desarrollaron una técnica de clasificación numérica para ayudar en la integración de software de gran tamaño. Utilizaron como criterio de agrupación la cantidad de conexiones entre una pareja de modulos, es decir la cantidad de términos que son comunes a dos módulos. Para un módulo modificado, el algoritmo utilizado construye una clasificación jerárquica donde las hojas son los módulos que pueden ser afectados por la modificación y los nodos agrupan los módulos que tienen un factor de conexión más elevado.

Los autores aplicaron su técnica al sistema Smile [Kaiser87]. A partir de dos módulos de este sistema (llamados CMDS y CMDDATA, fig 1 y 2) vastamente modificados, se construyó una matriz de similitud utilizando el código fuente de Smile. Las filas y las columnas representan los módulos del sistema Smile afectados por las modificaciones en CMDS y CMDDATA. El contenido de la matriz representa el número de objetos importados entre las parejas de módulos. Por ejemplo, entre los módulos UTIL y CMDS existen 17 términos intercambiados como lo muestra la figura 1. Un algoritmo de clasificación de aridad controlada [Maarek88a] utiliza esta matriz y produce un árbol de integración. La figura 2 [Maarek88a] muestra el árbol construido a partir de la aplicación de esta técnica.

El algoritmo utilizado forma grupos de manera ascendente, el primer nivel creado esta constituido por las hojas del árbol y cada una representa un módulo. Cada interacción produce un nuevo nivel, resultado de la combinación de una pareja de módulos, de nivel inferior más aquellos que presentan una fuerte similitud (es decir que tienen el mayor número de términos intercambiados). El proceso se repite hasta obtener el árbol. Para eliminar los

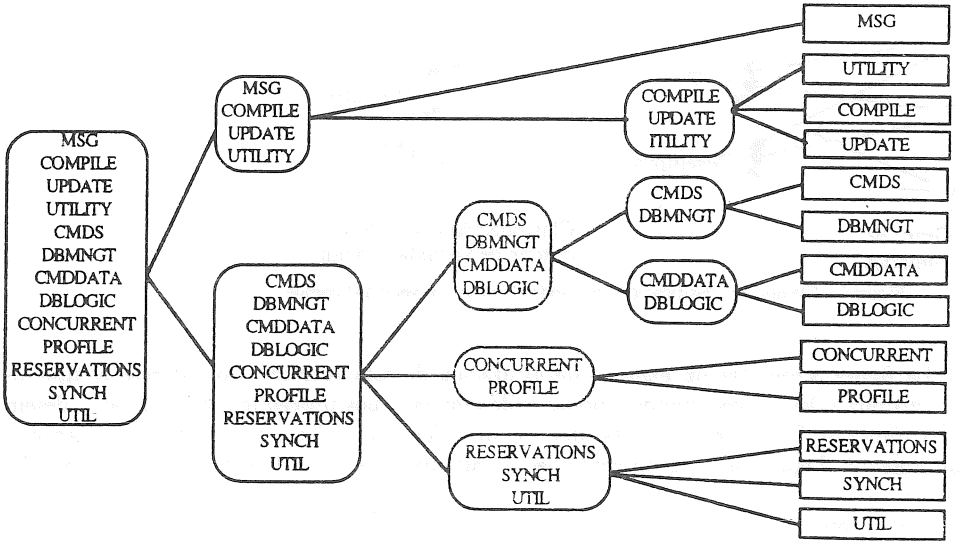


Figura 2 - Arbol de integración generado por el algoritmo de clasificación de aridad controlada.

grupos poco significativos, la dispersión estática de aridad de nuevos grupos es calculada en cada iteración por una función de varianza a partir de un valor ideal, parámetro suministradc por el usuario. La función de varianza ($\sigma$) esta definida por:

$$f_\sigma = \frac{\sum_{i=1}^{k}(x_i - \sigma)^2}{k}$$

donde:

$k$ = número de grupos de un nivel
$\sigma$ = valor ideal suministrado por el usuario
$x_i$ = número de descendentes del grupo $i$ en el último nivel.

En cada iteración, si el valor de $\sigma$ del nuevo nivel producido es mayor que el valor del nivel inferior, el nuevo nivel queda terminado. Al final del proceso el árbol producido será una colección de niveles terminados. Por definición, el primer nivel terminado constituye las hojas del árbol.

## 3.3 Restructuración de sistemas viejos

[Schwanke89a y 89b] utiliza una técnica de clasificación conceptual para extraer una estructura jerárquica del software y para compararla con la estructura proyectada. El criterio de clasificación utilizado se denomina criterio de "vecindad común". A partir de un grafo de llamados a procedimientos donde los nodos son los componentes del software y los arcos las referencias cruzadas. La técnica empleada reagrupa los nodos que tienen atributos comunes.

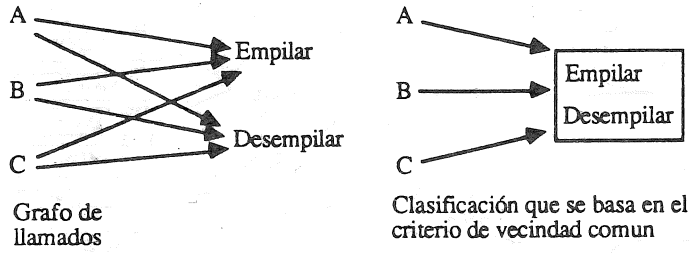


Figura 3 - Ejemplo de clasificación utilizando el criterio de vecindad común

La figura 3 muestra un ejemplo de utilización de esta técnica. Considerese el tipo abstracto pila y los procedimietos empilar y desempilar. Los procedimietos empilar y desempilar nunca se llaman entre si, asi que los módulos que utilizan una pila los llaman conjuntamente. Estos dos procedimientos serán agrupados si se utiliza el criterio de vecindad común.

Los autores utilizaron una técnica ascendente de clasificación jerárquica para poner en práctica este criterio. Esta técnica produce una descripción jerárquica del software en dos pasadas. La primera pasada construye pequeños grupos de alto grado de similitud, después los combina en grupos más grandes hasta formar un árbol de clasificación, donde las hojas son los componentes básicos del software y los subárboles los grupos. La segunda pasada elimina los subárboles de débil utilidad con el propósito de aumentar la aridad promedio y de disminur, por consequencia, la profundidad del árbol creado. La utilidad de un subárbol se mide con una función de utilidad de grupo (denotada FUG). La FUG es el producto del tamaño y de la "*pureza*" del grupo. La pureza es la probabilidad de asociación de un atributo a un miembro del grupo. Esta función favorece los grupos en los cuales la mayoríia de los miembros tienen muchos atributos en común.

La FUG utilizada por los autores es una adaptación del procedimiento de clasificación conceptual usado por el sistema Cobweb [Fischer87]. El sistema Cobweb produce un árbol de probabilidad a partir de una colección de objetos y donde la FUG es maximizada en cada nivel.

Los autores experimentaron su método sobre un software (representado por un grafo de 82 nodos y 155 arcos) descompuesto en 29 grupos. El tiempo de ejecución del algoritmo en una estación SUN de 12MB fue de 323 segundos.

## 3.4 Clasificación de software reutilizable

[Arango88a y 88b] utilizó el sistema Cobweb [Fischer87] para caracterizar las clases de software reutilizables. Se construyó un árbol probabilístico a partir de un conjunto de atributos que describen el software potencialmente reutilizable. Cada nodo de este árbol representa una clase de objetos que tienen características comunes. Estas características son formuladas por consultas. Una probabilidad es calculada para cada clase de objetos (representados por un nodo) teniendo en cuenta cada una de las características.

[Arango88b] aplicó la técnica sobre aplicaciones sobre un ambiente Unix. Las aplicaciones seleccionadas fueron descritas por 26 atributos. Se construyó una matriz de servicio donde las filas representaban las características resultado y las columnas el nombre de la aplicación.

| | Q-68 | Q-69 | Q-71 | Q-83 | . . . . | Q-111 |
|---|---|---|---|---|---|---|
| Maintaining-Record-Command-History | y | y | y | y | . . . | y |
| Renaming | y | y | y | y | . . . | y |
| Saving-Command-History-at-Logout | n | n | n | n | . . . | n |
| Executing-Commands-From-File | n | n | y | y | . . . | n |
| Acces-to-Bulletin-Boards | n | n | n | n | . . . | n |
| Checking-on-Mail | n | n | n | n | . . . | n |
| Enabling-Notify-End-Of-Jobs | n | n | n | n | . . . | n |
| . . . . . | . . . | . . . | . . . | . . . | . . . | . . . |
| Changing-Automatic-Timing-Command | n | n | n | n | . . . | n |

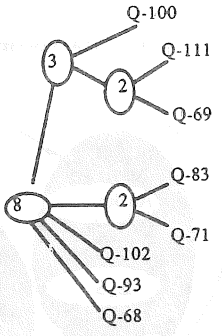


Figura 4 - Ejemplo de clasificación de aplicación utilizando el sistema Cobweb.

La figura 4 muestra una parte de la matriz construida. Por ejemplo, la primera columna, denotada Q-068, identifica el script .cshrc. Las funcionalidades de este script descritas en la matriz son: el mantenimiento de los registros de interacción del usuario con el sistema, los servicios de cambio de nombre... La matriz y la FUG del sistema cobweb fueron utilizadas para crear el árbol de clasificación de software reutilizable. Cada nodo del árbol representa un grupo de la aplicación que tienen caracteristicas parecidas.

## 3.5 Administración de bibliotecas de software reutilizable

[Maarek88b y 89b] desarrolló la herramienta GURU para construir automáticamente bibliotecas de software reutilizable. La herramienta permite:

- extraer los atributos significativos de la documentación del software utilizando una técnica de clasificación numérica;
- estructurar jerárquicamente las bibliotecas de código reutilizables utilizando una técnica de clasificación conceptual.

El proceso de extracción automática de atributos se hace en dos pasadas. En el transcurso de la primera pasada, los textos son analizados para encontrar las afinidades léxicas (denotadas AL) entre parejas de palabras. En la segunda pasada, los ALs significativos son seleccionados utilizando una técnica de clasificación numérica. Este algoritmo es análogo al utilizado por el módulo de particionamiento Cluster/2 1 [Michalski83]. El objetivo por alcanzar consiste en generar una cantida optimal de k-particiones sobre un conjunto de objetos, donde k es un parámetro establecido por el usuario. Este enfoque intenta reducir la cantidad de atributos producidos en la primera pasada y de identificar las clases de ALs más significativas.

La creación de bibliotecas se hace con un algoritmo de clasificación conceptual [Maarek89a] derivado de Unimen2 [Lebowitz86]. El algoritmo utiliza los componentes del software y sus atributos (gracias al proceso de indentación descrito antes). Esta técnica presenta las características siguientes:

- Hace parte de las técnicas de clasificación conceptuales llamadas jerárquicas. Identifica automáticamente las clases de una jerarquía;

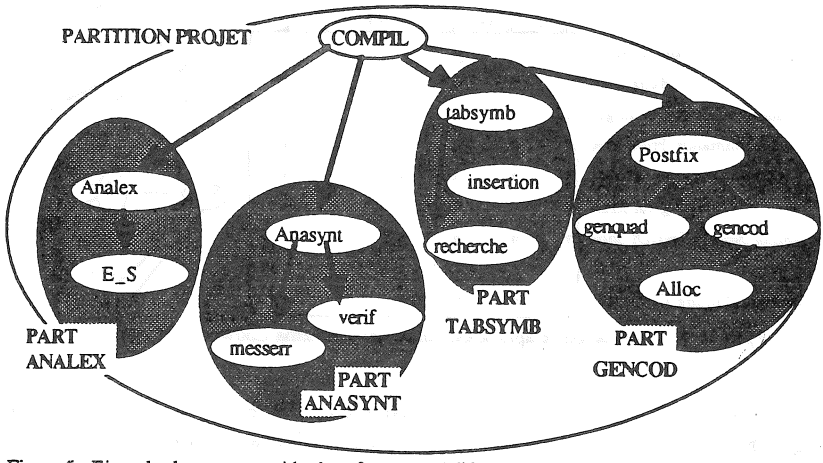


Figura 5 - Ejemplo de estructuración de software en Adèle

# 5 Aplicacion en el ambiente Adele

Adele es un ambiente de programación global que se centra sobre los aspectos de gestión de configuraciones, gestión de una base activa de programas y gestión de tareas.

Adele ofrece funcionalidades para soportar la evolución de los objetos, de configuraciones y versiones multiples. Estas funcionalidades son desarrolladas via un modelo de datos construido alrededor de un objeto compuesto central llamado *familia*. La noción de familia narra la evolución en versiones multiples de la noción clásica de módulo. Sin embargo, Adele ofrece una mayor libertad en la definición de la estructura intermódulos.

Diferentes mecanismos son usados por Adele para permitirle al usuario especificar las propiedades de los objetos bajo la forma de atributos y pora controlar la estructura del sistema de una manera compacta y coherente.

## 5.1 Estructura del software

Esta parte describe brevemente los mecanismos básicos para el modelaje de un sistema en el ambiente Adele . Para una descripción más completa ver [Belkhatir88 y 89].

El modelo Adele genera el software bajo la forma de una jerarquía de familias definiendo asi la noción de subsistema (llamadas particiones en Adele). En la figura 5 se muestra la descripción de una estructura clásica de un compilador bajo la forma de diferentes subparticiones de familias. Cada familia define separadamente sus interfaces y sus realizaciones con sus respectivas dependencias.

Los tipos predefinidos que existen en Adele ayudan a definir los objetos básicos de una familia. Estos tipos básicos tienen una semántica predefinida que permite invocar las herramientas. Estos tipos pueden ser extendidos por el usuario. La estructura del software es conservada automáticamente por las herramienta, por ejemplo, la extracción automática de dependencias. Otros tipos de elementos y de relaciones pueden ser definidas por el usuario.

## 5.2 Aplicación de los métodos de clasificación

La aplicación de las técnicas de clasificación en el ambiente Adele pueden verse en dos etapas de la vida del software y pueden considerarse como:

- un mecanismo de la estructura en el caso en el que las técnicas de clasificación se apliquen en la etapa de desarrollo;
- un mecanismo de verificación de la coherencia de la estructura en el caso en el que las técnicas de clasificación se apliquen en la etapa de mantenimiento.

### 5.2.1 Aplicación en la etapa de desarrollo

Las técnicas de clasificación ayudan a especificar la jerarquía de familias y a definir y organizar los subsistemas (particiones).

La experimentación llevada actualmente es desarrollada utilizando el método de [Schwanke89a y 89b] para producir la estructura inicial del software. Este método ha sido utilizado porque se basa en la utilización de las propiedades de los componentes del software, mecanismo que existe en Adele. Este método de clasificación produce el árbol de familias y de subsistemas potenciales.

### 5.2.2 Aplicación en la etapa de mantenimiento

EL software de tamaño considerable, como consecuencia de su gran tiempo de vida, se desestructura a raíz de los cambios efectuados. Son necesarios mecanismos para controlar los cambios que afectan la estructura durante el mantenimiento.

La aplicación de técnicas de clasificación permiten, en la etapa de mantenimiento, producir la estructura del software después de los cambios y confrontantarla con la inicial para detectar desviaciones y sugerir nueva estructuración. En esta etapa la técnica de clasificación puede verse como una herramienta de verificación de coherencia de la estructura del software que está en evolución.

# 6 Conclusion

El algoritmo de clasificación ha sido desarrollado y probado sobre la relación de dependencia entre módulos. La estructuración del software, con base únicamente en esta relación, es insuficiente. El trabajo actual pretende incorporar las propiedades atadas a cada uno de los módulos. Estamos interesados particularmente en las propiedades derivadas de las herramientas de medida. Este es otro aspecto importante en estudio: desarrollar herramientas de medida del software. Con este trabajo se podrá incorporar en la clasificación propiedades significativas como medidas de modularidad, de control de-fluxo [Adam89], etc.

# Referencias


[Adam89] Jean-Michel Adam
*Quality evaluation and user assistance in the context of ASPIS.*
Institut IMAG - Laboratoire de Génie Informatique, rapport de recherche, March 1989.

[Arango88a] Guillermo F. Arango & E. Teratsuji
*Notes on the Application of the COBWEB Clustering Function to the Identification of Patterns of Reuse.*
Technical Report ASE-RTM-87, ASE, Dept. of Information and Computer Science, University of California, Irvine, CA, July 1988.

[Arango88b] Guillermo F. Arango
*Domain Engineering for Software Reuse.*
Technical Report 88-27 (PhD Dissertation), Depto. of Information and Computer Science, University of California, Irvine, CA, 1988.

[Belkhatir88] Noureddine Belkhatir
*Nomade: un noyau d'environnement pour la programmation globale*
Institut National Polytechnique de Grenoble, Décembre 1988, Thèse du Doctorat.

[Belkhatir89] Noureddine Belkhatir et Jacky Estublier. "Un gestionnaire d'activités de programmation globale pour Nomade." In *Proc. of the Second Internacional Workshop Software Engineering & Its Applications.* Toulouse (France), Dec. 4-8, 1989. p. 703-716.

[Everitt80] B. S. Everitt
*Cluster Analysis, 2nd ed.* Heineman Educational Books, London, 1980.

[Fischer86] Douglas H. Fischer & Pat Langley
"Methods of conceptual clustering and their relation to numerical taxonomy."
In *Artificial Intelligence and Statistics*, W. Gale (Ed.), Reading, Addison-Wesley, 1986.

[Fischer87] Douglas H. Fischer
"Knowledge acquisition via incremental conceptual clustering."
*Machine Learning*, 2(2):139-72, 1987.

[Hutchens85] David H. Hutchens & Victor R. Basili
"System structure analysis: clustering with data bindings."
*IEEE Transactions on Software engineering*, SE-11(8):749-57, Aug. 1985.

[Hutchens87] David H. Hutchens
*Software Tools for Data Bindings Analysis.*
Technical Report, Depto. of Computer Science, Clemson University, Clemson, SC, 1987.

[Kaiser87] Gail E. Kaiser & Dewayne Perry
"Workspaces and experimental databases: automated suport for software maintenance and evolution."
In *Proc. Conference on Software Maintenance*, Austin, September 1987, p. 108-14

[Lebowitz86] M. Lebowitz
"Concept learning in a rich input domain: generalization-based memory."
In *Machine Learning: An Artificial Intelligence Approach*, R.S. Michalski, vol. 2, J.G. Carbonell e T.M. Mitchell (Eds.), Morgan Kaufman, Los Altos, CA, 1986. pp. 193-214.

[Maarek87] Yoelle S. Maarek & Gail E. Kaiser
"Using conceptual clustering for classifying reusable ada code."
In *Proc. of the ACM SIGAda International Conference*, Boston, MA, Dec. 1987. Issue of the Ada Letters, ACM Press. p. 208-15.

[Maarek88a] Yoelle S. Maarek & Gail E. Kaiser
"Change management for very large software systems."
In *Proc. of the Seventh Annual International Phoenix Conference on Computers and Communications*, Phoenix, AZ, USA, March 16-18 1988. p. 280-5. IEEE Computer Society Press, Washington, DC.

[Maarek88b] Yoelle S. Maarek
"On the use of cluster analysis for assisting maintenance of large software systems."
In *Proc. of the Third Israel Conference on Computer Systems and Software Engineering*, Tel Aviv, Israel, June 6-7 1988. p. 178-86. IEEE Computer Society Press, Washington, DC.

[Maarek89a] Yoelle S. Maarek
*Concept learning via conceptual clustering.*
Research Report 14743, IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY. June 1989.

[Maarek89b] Yoelle S. Maarek; Daniel M. Berry; Gail E. Kaiser
*Automatically generating software libraries without pre-encoded knowledge.*
Research Report 14990, IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY. Aug. 1989.

[Michalski80] Ryszard S. Michalski
"Knowledge acquisition through conceptual clustering: a theoretical framework and algorithm for partitioning data into conjunctive concepts."
In *International Journal of Policy Analysis and Information Systems*, 4(3):219-43, 1980.

[Michalski83] Ryszard S. Michalski & Robert E. Stepp
"Automated constructions of classifications: conceptual clustering versus numerical taxonomy."
In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(4):396-409, 1983.

[Michalski87] Ryszard S. Michalski & Robert E. Stepp
"Learning from observation: conceptual clustering."
In *Machine Learning: an artificial intelligence approach*, Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell (Eds),Tioga, 1987. pp. 331-63.

[Newbery89] Frances Newbery
"Edge concentration: a method for clustering directed graphs."
In *Proc. of the Second Internacional Worksohop on Software Version and Configuration Control*. Princeton, Oct. 24-27, 1989. In *ACM SIGSOFT Software Engineering Notes*, 17(7):76-85, Nov. 1989.

[Pressman87] Roger S. Pressman
*Software Engineering: a practitioner's approach.* New York, McGraw-Hill, 1987.

[Schwanke89a] Robert W. Schwanke; R. Altucher; Michael A. Platoff
"Discovering, visualizing, and controlling software structure."
*ACM SIGSOFT Engineering Notes*, 14(3):147-50, Pittsburg, May 1989.

[Schwanke89b] Robert W. Schwanke & Michael A. Platoff
"Cross references are features."
In *Proc. of the Second International Workshop on Software Version and Configuration Control.* Princeton, Oct. 24-27, 1989. In *ACM SIGSOFT Software Engineering Notes*, 17(7):86-95, Nov. 1989.

[Selby88a] Richard W. Selby & Victor R. Basili
*Analyzing error-prone system coupling and cohesion.*
Technical Report, Dept. of Information and Computer Science, University of California, Irvine, CA, 1988.

[Selby88b] Richard W. Selby & Victor R. Basili
"Error localization during software maintenance: generating hierarchical system descriptions from the source code alone."
In *Proc. of the International Conference on Software Maintenance*, Phoenix, October 24-27, 1988, p. 192-7.

[Sokal63] Robert R. Sokal & P. H. A. Sneath
*Numerical Taxonomy.* Z. H. Freeman, San Francisco, 1963.

[Sokal88] Robert R. Sokal
"Unsolved problems in numerical taxonomy."
In *Classification and Related Methods of Data Analysis*, H. H. Bock (Ed.). Elsevier, North-Holland, 1988.