Revista Internacional de Métodos Numéricos para Cálculo y Diseño en Ingeniería. Vol. 11,3, 423–449(1995)

LA PROGRAMACIÓN ORIENTADA AL OBJETO APLICADA AL CÁLCULO POR EL MÉTODO DE LOS ELEMENTOS FINITOS

MIGUEL ÁNGEL BRETONES
ANTONIO RODRÍGUEZ FERRAN
y
ANTONIO HUERTA

Departamento de Matemática Aplicada III
E.T.S. de Ingenieros de Caminos
Universidad Politécnica de Cataluña
Campus Norte C–2, E–08034 Barcelona, España

RESUMEN

Los códigos orientados al objeto han surgido recientemente como una alternativa a los programas convencionales de cálculo por el método de los elementos finitos. La información se almacena según objetos de naturaleza más compleja que las variables habituales en FORTRAN u otros lenguajes. Estos objetos pueden ser combinados entre ellos para generar nuevos objetos, avanzando de esta manera en el proceso general de cálculo. En consecuencia, todo problema se resuelve con una sucesión de instrucciones de carácter notablemente abstracto.

La facilidad de acceso a la información, así como la capacidad de gestionar de manera genérica estructuras de datos muy elaboradas, convierten a la programación orientada al objeto en una opción muy válida en ingeniería y desarrollo. Algunas de sus posibilidades y características se muestran aquí, tanto desde un punto de vista formal como a través de diversos ejemplos de aplicación a diferentes problemas, resueltos con el código orientado al objeto CASTEM2000.

SUMMARY

Object–Oriented Programming is a new trend in software design and development which arises from the need of reducing the costs of creating and maintaining complex computer software. Much of the power of conventional programming comes from the abstract representation of data structures. Object–oriented programming takes a step further and represents everything as objects which may contain not only data, but also the functions which operate on that data. Thus, the basic philosophy of the object–oriented approach is to build parts of a large program as abstract "objects".

This new approach is being applied to the development of CASTEM2000, a FEM package for thermo-mechanical analysis. In this article, we present first, the theoretical basis of this ongoing effort. And second, we demonstrate the current capabilities of CASTEM2000 with a sample of selected examples.

Recibido: Octubre 1994

 ISSN 0213–1315

Palabras clave

Códigos orientados al objeto, programación orientada al objeto, método de los elementos finitos, cálculo de estructuras.

INTRODUCCIÓN

Evolución histórica

A lo largo de las últimas décadas y en el ámbito de la ingeniería civil, se puede constatar la creciente complejidad de los problemas que son abordados de manera cotidiana. Este fenómeno nace fundamentalmente de la necesidad de aproximar cada vez más entre sí la realidad y la idealización que de ella realizamos; el fin último perseguido es la limitación de incertidumbres –por ejemplo por lo que respecta al comportamiento estructural de las construcciones– con las importantes implicaciones económicas que esto conlleva. Dicha complejidad exige desde el punto de vista de los métodos numéricos por un lado, contar con herramientas de cálculo cada vez más potentes, y por otro, disponer de programas y estrategias de análisis capaces de afrontar de manera eficiente situaciones cada vez más cercanas a la realidad.

Paralelamente a esta circunstancia, el método de los elementos finitos (MEF) se ha impuesto como una técnica de extraordinaria versatilidad y potencia en la resolución numérica de problemas de ingeniería, siendo en el campo del cálculo estructural donde probablemente más se haya extendido. El auge de este procedimiento creció paralelamente con las posibilidades de cálculo de los ordenadores, pasando rápidamente de ser un recurso reservado para situaciones excepcionales a convertirse en imprescindible a la hora de abordar problemas de cierta entidad. En la actualidad nos encontramos en un momento en el que el MEF comienza a abandonar el terreno del análisis de problemas singulares –donde fue valorado desde su aparición– para pasar a convertirse en un procedimiento habitual de cálculo en situaciones ordinarias.

De modo consustancial a su propio desarrollo surgió la necesidad de implementar el MEF, apareciendo de este modo –y sobretodo a lo largo de los últimos años– una gran cantidad de paquetes comerciales de programas, fundamentalmente dedicados al cálculo de sólidos y estructuras. Frecuentemente, estos programas están escritos en FORTRAN, C o en cualquier otro de los lenguajes de programación compilados conocidos como de "alto nivel".

Situación actual

Los códigos de programación tradicional, al menos los antecesores de los actuales, son lógicamente contemporáneos a la aparición de los primeros ordenadores. Quizás por ello trabajan con estructuras de datos simples y con una lógica de operación notablemente reducida: la unidad mínima –y fundamental– de información es la variable, que representa un valor escalar o alfanumérico asignado a una determinada posición de memoria[1].

En el estado actual de desarrollo de la mecánica computacional, dichos códigos adolecen de una seria problemática derivada de la situación descrita anteriormente:

presentan graves dificultades a la hora de efectuar modificaciones que amplíen sus posibilidades o que mejoren su modelización de la realidad.

Asumiendo el hecho de que el empleo de subrutinas permite conferir una cierta estructura modular a los programas, facilitando eventuales modificaciones –a cambio, eso sí, de prestar una especial atención al paso de la información entre los diversos módulos– las dificultades para llevarlas a cabo radican en la propia esencia informática de estos códigos: por una parte, la simplicidad de las estructuras de datos que antes aludíamos, y por otra, la secuencialidad de las acciones que ejecutan[2] (Tabla I).

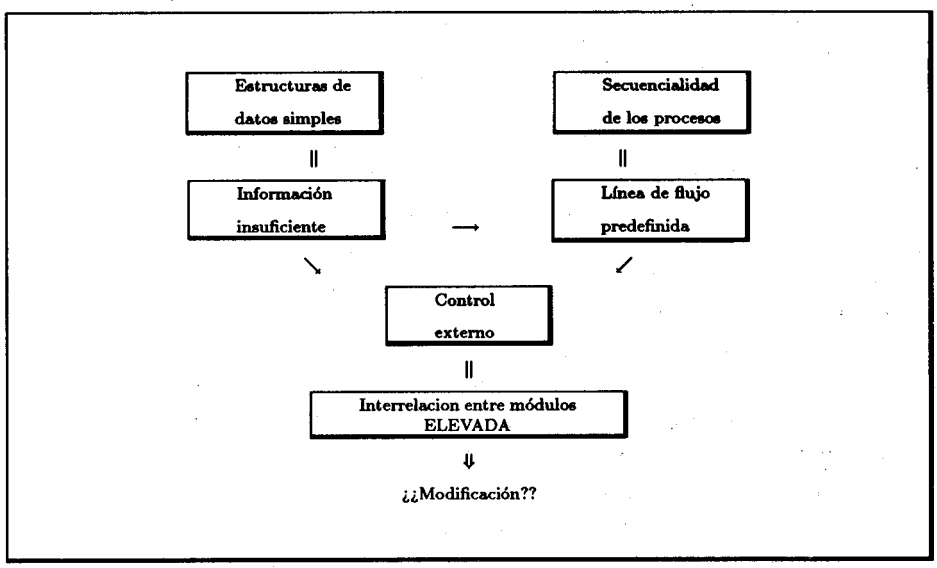


Tabla I. El problema de los códigos tradicionales, función–orientados

Así un determinado módulo o procedimiento –una subrutina en el caso de los lenguajes convencionales– siempre realiza una tarea concreta, y lo que es más importante, siempre la misma: los argumentos de todo módulo serán un determinado conjunto de variables, obteniendo como resultado otro conjunto –igual o diferente– de variables; naturalmente, estas últimas o más concretamente sus valores, dependerán de los valores iniciales que tomen las variables de entrada. Con todo, los datos de partida siempre serán transformados de la misma manera, esto es, siguiendo las mismas pautas y con independencia de su estado original. Desde este punto de vista, se trata de un procedimiento en absoluto adaptable a las posibles características específicas de la información de entrada.

Así una subrutina clásica de resolución de sistemas lineales de ecuaciones tiene como entrada de datos un conjunto de valores reales que conforman una matriz de coeficientes y un vector de incógnitas. No obstante, ello no quiere decir que los argumentos del módulo sean la matriz y el vector, sino tan sólo un conjunto de escalares a los que nosotros conferimos un determinado significado.

Simultáneamente, dichas variables –el único tipo de argumento relevante en un módulo convencional– no aportan una cantidad de información autosuficiente; de hecho, puede afirmarse que la única funcionalidad de una variable es el acceso a su valor

concreto. Los datos que almacenan de manera individualizada no son suficientes como para caracterizar un determinado objeto, entendiendo como tal los conceptos abstractos que manejamos a la hora de resolver un problema; así por ejemplo una matriz de conectividades no es una malla de elementos finitos. Este hecho implica automáticamente una dependencia funcional entre las diversas variables presentes en un programa.

En consecuencia, si los procesos de transformación están prescritos *a priori* y las variables carecen de significado autónomo, se puede afirmar que cualquier módulo maneja datos sin realmente controlarlos. Así dicho control deberá realizarse externamente a través de conjuntos de instrucciones –subrutinas– específicas y únicamente destinadas a tal efecto. Para grandes códigos de cálculo esto último no resultará fácil ni cómodo; estas son las limitaciones clave a la hora de buscar la modularidad y la generalidad de los procedimientos en los códigos convencionales.

Por otra parte, el carácter compilado del lenguaje de programación, tan útil durante la fase de ejecución, entorpece la etapa de desarrollo al tener que recurrir a la integración –a nivel de lenguaje fuente– de la correspondiente ampliación con el resto del código. Este procedimiento siempre deberá realizarse a la hora de añadir –y eventualmente verificar– cualquier mejora.

En un intento de salvar esta situación, muchos de estos códigos suelen ofrecer puntos de entrada, previstos por sus programadores, con objeto de facilitar incorporaciones de nuevos módulos. No obstante, incluso a pesar de que pueda disponerse de un protocolo exhaustivo de modificación –por ejemplo por lo que a manejo de variables ya existentes se refiere– ello puede devenir en una tarea imposible. Fundamentalmente esto es debido a dos factores: por un lado, la línea de flujo del programa se encuentra predefinida y en la mayoría de casos no resultará fácil alterarla sin modificar otras zonas del programa, y por otro, el nivel de interrelación de unos módulos con otros es demasiado profundo. De este modo, y debido al control externo de los datos a que antes hacíamos referencia, un error cometido en un nuevo módulo puede determinar no sólo la incompatibilidad de éste con el resto del programa, sino generar errores graves de cálculo que además pueden manifestarse en otras partes del código.

De todo lo anterior se deduce que la modificación de uno de estos programas puede exigir el conocimiento amplio, profundo y exhaustivo de su totalidad, no pudiendo circunscribirse al aspecto concreto que se desee mejorar. Este hecho resta notable eficiencia el sistema de programación convencional, puesto que necesita, desde este punto de vista, de un gran esfuerzo inicial para producir resultados. Además, este hipotético proceso de evolución imposibilita el desarrollo paralelo del mismo código por parte de grupos de trabajo diferentes.

Con todo, una ventaja de la concepción habitual de estos lenguajes –función orientados– radica en el hecho que, al menos en el rango de aplicación de los mismos, su eficiencia en términos de tiempo de computación puede ser notablemente elevada: las estructuras de datos y los algoritmos de programación pueden ser suficientemente específicos como para incluir técnicas muy elaboradas de optimización de costos de memoria y CPU. De todos modos, ello reducirá todavía más las posibilidades efectivas de ampliación de los mismos, si se pretende que tanto los algoritmos como las estructuras de datos tengan cierta vocación de generalidad.

CÓDIGOS ORIENTADOS AL OBJETO

En un intento de superar las limitaciones de los programas convencionales, han aparecido recientemente los primeros códigos orientados al objeto (COO) en el ámbito de los elementos finitos. La finalidad que persiguen es, en definitiva, dotar de una mayor comodidad y flexibilidad a todas las acciones encaminadas a su manejo, tanto desde el punto de vista del usuario como para el programador encargado de su desarrollo. Se trata, por tanto, de ampliar al máximo el rango de problemas que pueden abordarse, diseñando una estructura de programa y un sistema de gestión de la información que facilite su rápida y eficiente modificación[3,4].

Conceptos fundamentales: objetos, operadores y clases

Los COO –también llamados programas orientados al dato, por oposición a los tradicionales, orientados a la función– se caracterizan por la manipulación de la información según estructuras de datos de naturaleza más compleja que las propias de los lenguajes convencionales. La filosofía del sistema nace de la idea que mayor complejidad significa también más información sobre la esencia misma de aquello que se pretende representar, facilitando de esta manera su posterior manejo.

En consecuencia, los llamados "objetos", integrantes de los COO, representan las entidades y conceptos abstractos requeridos para la resolución de problemas. Así en un cálculo estructural por elementos finitos, interesará trabajar con nodos, mallas, campos escalares o vectoriales definidos sobre los nodos (desplazamientos, cargas) o en los elementos (tensiones, deformaciones), matrices de rigidez y de masa, funciones de forma, interpolaciones de diversos grados etc. En un COO por ejemplo, una malla de elementos finitos puede ser un objeto en sí mismo, en lugar de una matriz de coordenadas nodales y una matriz de conectividades. De esta manera la malla es un objeto directamente manejable, que podrá ser representado gráficamente, deformado según un campo de desplazamientos, o ser soporte de una cierta formulación de elementos finitos (tensión o deformación plana, elementos de viga, de placa etc.). Fundamentalmente ello es posible porque toda la información necesaria para caracterizar el "objeto malla" no se encuentra fragmentada a lo largo de diversos módulos, lo que a su vez posibilita la independencia –ortogonalidad– de unos objetos con otros.

De modo más abstracto, podemos definir un objeto como un conjunto estructurado, completo y autónomo de datos que caracterizan de manera íntegra un concepto geométrico o matemático. Así la resolución de un problema se reduce al manejo adecuado de estos objetos; los resultados de la manipulación de objetos preexistentes –con vistas a generar nuevos objetos– van constituyendo las secuencias elementales de cálculo, propias del método empleado (MEF en nuestro caso) y no del problema concreto a tratar: generación de malla, elección de la formulación y de los parámetros de los materiales, definición de las cargas y de las vinculaciones, cálculo de las matrices de rigidez, obtención de los desplazamientos, postproceso etc. La manera concreta en que estos objetos son manipulados para la producción de nuevos objetos, las posibilidades de combinación y tipos de resultados alcanzables así como eventuales incompatibilidades,

pueden estar ya incorporados como parte de la esencia que define al objeto[5], o pueden ser especificadas a *posteriori* a través del concepto de «operador».

En los COO se pueden emplear operadores para representar las secuencias elementales a que antes hacíamos referencia; todo operador realiza una serie de transformaciones simples capaces de generar, a partir de uno o varios objetos preexistentes otro nuevo, avanzando así en el proceso de resolución del problema. De esta manera la programación tiende a ser más abstracta que en los lenguajes tradicionales. En éstos, el programador tiende a pensar en términos de las acciones concretas que debe realizar para generar un determinado procedimiento; como contrapunto, en un COO se piensa en términos de las acciones conceptuales que requiere un determinado problema de programación (ensamblar dos matrices, resolver un sistema de ecuaciones, cargar una estructura, dotarla de vinculaciones etc.) para ser resuelto satisfactoriamente[6] (Tabla II).

El problema: Disponer de un elemento finito multicapa para el análisis de placas de hormigón armado.

El punto de vista de la programación tradicional:

Objetivo: definición del elemento multicapa para su posterior incorporación a un programa tradicional de MEF.

Caracterización: El nuevo elemento vendrá caracterizado por su soporte geométrico, grado y tipo de interpolación, así como por los modelos reológicos de cada capa, con sus correspondientes puntos de integración de tensiones. Cada uno de estos grandes grupos de propiedades irá asociado a una subrutina específica en el proceso de construcción del nuevo elemento.

Procedimiento:

- Generación de la subrutina que crea el soporte geométrico, disposición de los nodos y puntos de integración, numeración local y global ...
- Subrutina de generación de las funciones de forma y matrices asociadas.
- Subrutina de cálculo de la matriz constitutiva a partir de las propiedades y disposición de cada una de las capas.
- Subrutina de obtención de la matriz de rigidez global del elemento.
- Subrutina de cálculo de fuerzas nodales equivalentes.
- Subrutina de cálculo de tensiones elementales a partir de un campo de desplazamientos.
- La suma de todos los procesos anteriores define el nuevo elemento. En la medida en que puedan existir en el código otros elementos similares a éste, algunos de estos pasos podrán obtenerse como generalización de lo ya existente (exceptuando tal vez los pasos 3 y 4)

El punto de vista de los códigos orientados al objeto:

Objetivo: obtener un nuevo objeto –elemento finito placa multicapa– a partir de otros ya presentes en el código –por ejemplo elementos finitos de placa simples–.

Caracterización: reunirá las características propias de los elementos que lo integren —en este caso elementos finitos monocapa— así como las derivadas de la manera concreta de disponerlos. A la vez, cada elemento monocapa seguirá un proceso de caracterización homólogo a partir de objetos más simples.

Procedimiento:

- Se escoge el objeto–soporte geométrico; en este caso particular cuadrilátero de cuatro nodos.
- Se definen dos objetos–modelos de comportamiento asociados respectivamente al hormigón y al acero.
- Cada uno de los objetos–elementos finitos monocapa nace de la reunión del soporte geométrico, el modelo reológico, el grado de interpolación y la excentricidad de la capa respecto al plano medio de la placa (En definitiva objetos y atributos de objeto).
- El objeto final es la fusión del número necesario —uno por cada capa a representar—de elementos generados por los procesos anteriores.
- El nuevo objeto pasa a ser una entidad autónoma, independiente de los objetos intermedios empleados y del proceso seguido para su generación.

Tabla II. El concepto de programación orientada al objeto frente a la concepción tradicional

Los objetos a su vez pueden agruparse en "clases" que reúnen las características comunes a todos aquellos pertenecientes a la misma. Así, por ejemplo, podríamos tener la clase de las matrices, de los campos escalares o vectoriales definidos en los elementos o en los nodos etc. Desde este punto de vista, cada objeto concreto puede ser visto como "derivado" de su correspondiente clase; posteriormente veremos bajo qué reglas se produce este fenómeno.

Características definitorias

La programación orientada al objeto (POO) se distingue en primer lugar por la posibilidad de usar y representar conceptos abstractos. Los atributos de un objeto son las características esenciales que lo definen. Éstos sólo pueden ser modificados –para producir un nuevo objeto– bien a través del propio objeto (si en sus atributos se han incorporado dichas posibilidades de modificación), bien por un operador asociado inequívocamente a él.

En cualquiera de los dos casos, toda la información requerida para llevar a cabo cualquier manipulación está totalmente contenida en el propio objeto o, a lo sumo, en el binomio operador–objeto. Este hecho es el que se conoce como "encapsulación de la información". Por ejemplo, todos los datos necesarios para transformar una malla de elementos finitos formada por cuadriláteros en otra que contenga triángulos –configuración, líneas de contorno, conectividades, numeraciones de nodos y elementos etc.– está disponible como parte integrante del "objeto malla". Esta característica se hace extensiva a cualquier transformación de cualquier naturaleza que pueda experimentar una clase presente en el código.

También ha quedado reflejado cómo un objeto se genera como modificación de otro/s preexistente/s, incorporando la parte de información que a ambos es común; este mecanismo es conocido por el nombre de "herencia". Así un campo de características mecánicas generado para el análisis de un sólido (nuevo objeto), automáticamente incorpora información acerca de la geometría soporte del problema (viejo objeto) así como del modelo de comportamiento al que va asociado (viejo objeto). Este concepto también suele asociarse a la información que recibe un objeto como perteneciente a una determinada clase, evitando de esta manera duplicidades innecesarias de información.

El fenómeno de ortogonalidad entre objetos descrito anteriormente resulta complementario del principio de no anticipación, también característico de la POO. Según él, la generación de un objeto no puede depender de su posterior utilización; sólo de esta manera se garantiza completamente la independencia de los diferentes objetos.

Finalmente, la última gran característica definitoria de la POO es conocida como "poliformismo" o "polisemia"; se trata de la posibilidad de que una determinada entidad presente en el código (preferentemente un operador) pueda adoptar diversas formas. Con ello se consigue que objetos de diferentes clases puedan responder de manera distinta a la misma operación. Por ejemplo, en un determinado COO podrá existir un operador llamado '+' encargado de fusionar objetos. Esa idea abstracta de fusión se traducirá en una suma algebraica cuando se aplique a dos escalares, pero también representa el ensamblado de dos matrices de rigidez o la superposición de dos campos nodales, definidos sobre una geometría común.

Propiedades

Las consecuencias que se derivan de esta estructura de datos jerarquizada en objetos manejados según operadores son de índole tanto conceptual como práctico. En primer lugar, cada nueva combinación de secuencias operador–objeto puede –potencialmente– generar un nuevo resultado desde el punto de vista teórico, esto es, puede dar respuesta a un nuevo problema. Así las posibilidades de adaptación del COO a la resolución de problemas sólo quedarán limitadas por el número y la naturaleza de los objetos y operadores presentes en el mismo capaces de ser generados a partir de los ya existentes.

Este hecho, que constituye gran parte de la potencia del método, entraña a su vez gran peligro: la aplicabilidad y validez de las combinaciones de objetos vendrá además impuesta por los límites teóricos del MEF. Por esta razón es imprescindible conocer el fundamento teórico de las acciones que pretendemos llevar a cabo, a pesar de que para realizarlas a través de un COO nunca sea necesario el "programarlas" en el sentido clásico del término.

Al tratarse de estructuras de datos independientes, la adición de nuevos objetos –a través del mecanismo de herencia y por encima del nivel básico de compilación– no deberá afectar a sus antecesores, aunque sí puede hacerlo la incorporación o modificación de clases.

Por otra parte, es viable el desarrollo paralelo –pero a la vez conjunto– de un COO por parte de diversos grupos de trabajo, por no afectar su evolución a la parte de código inicialmente generada.

La incorporación de una nueva clase o de un objeto original –sin herencia completa de otros preexistentes– tampoco obligaría a la reescritura íntegra del código, sino sólo de las estructuras de datos relacionados con el mismo. Nos encontraríamos así en una situación parcialmente semejante al caso de la programación tradicional, con la diferencia de que en POO ésta representa un segundo y más profundo nivel de evolución, pudiéndose lograr notables avances sin tener que recurrir a él, esto es, utilizando los mecanismos de evolución y herencia aquí descritos.

Finalmente, debe tenerse en cuenta que la constante utilización de estructuras complejas de datos, así como el recurso a operadores de actuación genérica –polimórfica– para llevar a cabo su manipulación, afectarán a la eficiencia en costo de computación del código.

CASTEM2000, UN CÓDIGO ORIENTADO AL OBJETO

A lo largo de los dos últimos años los autores del presente artículo han trabajado con el software CASTEM2000. Se trata de un COO destinado al cálculo generalizado de sólidos y estructuras por el MEF, creado por el Comisariado de la Energía Atómica (C.E.A.) francés.

Su configuración actual incluye gran variedad de clases de objetos así como de operadores –unos 300–. En la actualidad y tras distintos acuerdos de colaboración científica, diversos grupos de trabajo europeos colaboran en su desarrollo.

La finalidad de CASTEM2000 es proporcionar un entorno de trabajo, semejante por ejemplo al ambiente UNIX, en el que pueden llevarse a cabo todos los cálculos

propios de los problemas que trata de abordar. Esto quiere decir que dos etapas tradicionalmente separadas del cálculo según el MEF como son el pre y postproceso se encuentran incorporadas a él, no distinguiéndose esencialmente de la secuencia de operaciones que conforman la resolución efectiva de problema.

Características específicas

Desde un punto de vista formal, CASTEM2000 se caracteriza por diluir el concepto de clase, de tal manera que los objetos se generan unos a partir de otros, pero no derivan –ni siquiera los más elementales– explícitamente de una supraentidad más abstracta, la clase. Ello no invalida en absoluto la clasificación de CASTEM2000 como un COO, sino que confiere una mayor independencia, si cabe, al conjunto de objetos, a cambio de interrumpir la cadena de generación de éstos en base a esa entidad más genérica que el propio objeto. Asimismo el concepto de operador está permanentemente presente en su funcionamiento: los objetos no incorporan como información de base posibilidad alguna de ser modificados, dejando este cometido en exclusiva para los operadores implementados en el código. Se consigue así flexibilizar la estructura de relación entre un objeto primario y aquellos que derivan de él, que sólo dependerá del operador concreto empleado –presente en el código original o elaborado de nuevo– pero sin afectar a la información presente en el mismo.

De la misma manera que se generan unos objetos a partir de otros gracias al mecanismo de herencia, otro tanto sucede con los operadores: en el entorno de CASTEM2000 es factible definir secuencias ordenadas de instrucciones –conteniendo diversos operadores– de modo que, a partir de un conjunto prefijado de clases de objetos, obtengan un nuevo conjunto de objetos. En esta situación, el código es capaz de reconocer dicha secuencia como si de un nuevo operador –metaoperador desde el punto de vista de contener operadores "simples" en su definición– se tratara. Naturalmente, el fenómeno de herencia se manifiesta entre este nuevo metaoperador y sus antecedentes (Tabla III). Así a partir del operador capaz de resolver sistemas de ecuaciones lineales pueden construirse toda una familia de nuevos metaoperadores capaces –por ejemplo– de abordar de manera incremental problemas no lineales.

CASTEM2000 se apoya en los principios de ortogonalidad y no anticipación anteriormente expuestos. Ello conduce, por ejemplo, a que la implementación de condiciones de contorno de tipo Dirichlet no se realice de la manera habitual –esto es, modificando los coeficientes de la matriz asociada al sistema de ecuaciones a resolver– sino utilizando la técnica de los multiplicadores de Lagrange. De esta manera se consigue que la generación de las matrices de rigidez no contenga ninguna restricción *a priori* (principio de no anticipación) y –lo que es más importante– no incorpore ninguna hipótesis previa acerca, por ejemplo, de la naturaleza del método de resolución: con una misma matriz elemental y diversas condiciones de contorno, generadas *a posteriori* y ensambladas con la rigidez original, podremos obtener resoluciones bien en desplazamientos, bien en esfuerzos etc.

CASTEM2000 puede ser visto, en suma, como una base de datos de objetos nominados y tipificados, que pueden ser manejados de manera dinámica a través de un conjunto elemental de operadores también tipificados. Las secuencias de instrucciones conteniendo operadores y objetos, sus reglas de combinación y sintaxis, conforman

el GIBIANE, que es *de facto* un lenguaje de programación de alto nivel, basado en los conceptos abstractos propios de la POO[7]. Asimismo el carácter precompilado de CASTEM2000 hace posible su utilización de manera interactiva. Visto de otro modo, las instrucciones en GIBIANE son ejecutadas mediante la intermediación de un intérprete; es por este motivo que CASTEM2000 puede ser entendido como un entorno de programación.

Por otra parte, la documentación referente a los recursos al alcance del usuario y al estado concreto de evolución del código se haya permanentemente disponible también de manera interactiva, facilitando de esta manera la fase de diseño de nuevos desarrollos[8].

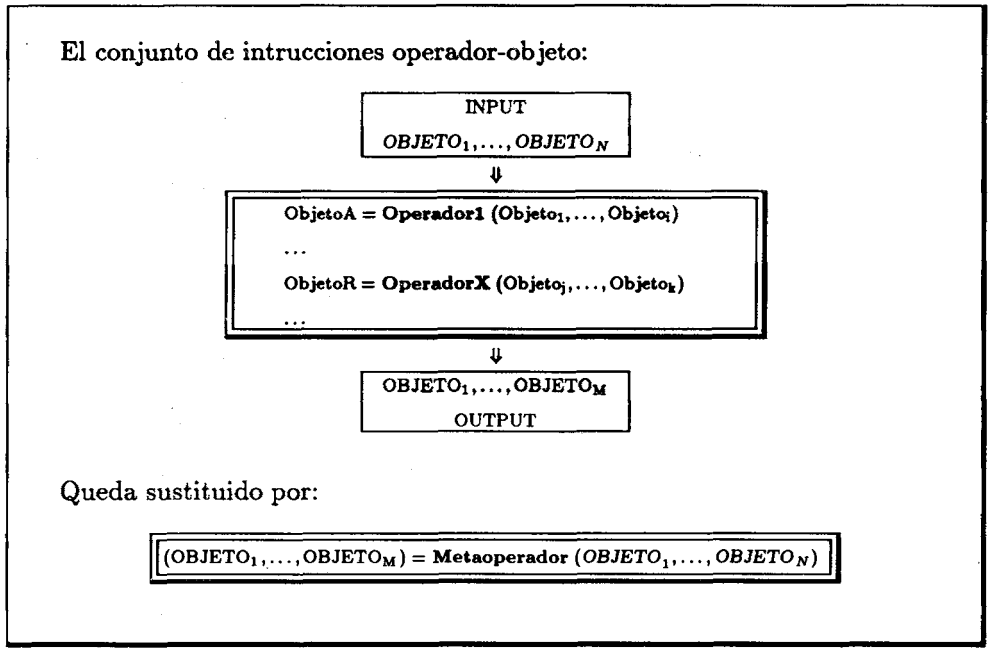


Tabla III. Definición de un metaoperador a partir de otros ya existentes. Se manifiesta el fenómeno de herencia

Niveles de utilización

A la vista de todo lo anteriormente comentado, y fundamentalmente a causa de las diversas estructuras superpuestas de organización y gestión de la información presente en él, CASTEM2000 permite su uso a muy diversos niveles. Éstos podrían ser, en orden creciente de complejidad, los siguientes:

- Disponiendo de un paquete verificado de instrucciones en GIBIANE, puede ser empleado para la resolución de un problema concreto, dejando libertad al operador para alterar únicamente los parámetros mecánicos o geométricos que lo gobiernan.

- En segundo lugar está lo que podríamos considerar la auténtica programación en GIBIANE, que posibilita la resolución problemas termo–mecánicos genéricos combinando operadores y objetos.

- Otra posibilidad, muy ligada a la anterior, consiste en la elaboración de metaoperadores, que extienden las posibilidades de generación de nuevos objetos.

- El último nivel, antes de lo que puede considerarse como el estado primitivo del código, lo constituye la utilización de su estructura e implementación del MEF para afrontar problemas inicialmente no previstos. Así se puede recurrir a la generación de objetos con lo que podríamos llamar una herencia "parcial" de sus antecedentes, incorporando información de base adicional, siempre por encima del nivel básico de compilación, esto es, producida a partir de objetos y operadores de carácter muy general, conocidos en otros códigos como "constructores". Un claro ejemplo lo constituye el estudio por el MEF de la propagación de ondas de sonido mediante el uso de CASTEM2000. Ello nos conduce a tener que resolver la ecuación de Helmholtz, cuya matriz de rigidez guarda similitudes con la procedente de un problema estructural; partiendo de lo ya existente y transformando de manera adecuada los objetos afectados –por ejemplo con la adición de nuevas matrices de rigidez generadas de manera semimanual– puede conseguirse el resultado perseguido[9].

- Finalmente, queda el desarrollo a nivel del código fuente, que posibilita la incorporación de nuevas clases de objetos y operadores, sin ninguna restricción *a priori*. Se trata de la situación más semejante a la propia de la programación tradicional, con la diferencia de que en ese caso representa la única vía de evolución, siendo en POO el último recurso de una serie de niveles intermedios de gran potencia. Otra aplicación de este nivel de programación puede constituirlo la transformación de procesos tipificados –por ejemplo metaoperadores– en operadores propiamente dichos en busca de una mayor eficiencia numérica.

RESOLUCIÓN DE DIVERSOS PROBLEMAS DE INGENIERÍA

A continuación se presentan los resultados correspondientes a la implementación en CASTEM2000 de diferentes problemas en el ámbito de la ingeniería civil[10,11]. El criterio seguido para su selección nace de la voluntad de que cada uno de ellos sirva para poner de manifiesto algunas de las características definitorias de la POO.

Estructuras de hormigón armado

En primer lugar, ilustraremos el estudio de estructuras de edificación compuestas por pilares y forjados. El objetivo perseguido es analizar el comportamiento de dichas estructuras, en régimen estático, según estados de carga lo más arbitrarios posibles.

Desde el punto de vista de la programación convencional, resultaría costos generar un algortimo que permitiera definir campos de carga no uniformes de manera sencilla,

en el sentido de resultar fáciles de manejar para un eventual usuario del código, distinto a su programador. En nuestro caso, la dificultad se salva individualizando cada uno de los objetos "carga" asociados a los estados que se consideran como básicos; de esta manera se definen, como entidades autónomas –objetos– las cargas sobre recuadros, voladizos o pilares, en cualquier dirección del espacio, resultando todo estado final de acciones como combinación lineal de las mismas. En consecuencia, no sólo la flexibilidad del procedimiento es muy elevada, sino también su sencillez, dado que el operador sólo debe introducir los coeficientes (positivos, negativos o nulos) de la citada combinación lineal. Este procedimiento no es exclusivo de la POO, lo que sí es original, es su economía, en base a la naturaleza y funcionalidad de los objetos que ésta incorpora. Así la instrucción en GIBIANE

$$\texttt{camporj} = \texttt{coefr.1} * \texttt{campr.1} + \texttt{coefr.2} * \texttt{campr.2} + \texttt{coefp.1} * \texttt{campp.1}$$

materializa la citada combinación lineal de acciones. `campr.1` y `campr.2` representan a los objetos–campo de carga unitario asociados a dos recuadros del forjado; por tanto, se trata de objetos esencialmente distintos, dado que estarán referidos a mallas de elementos finitos disjuntas, que incluso pueden soportar modelos de comportamiento mecánico diferentes. Asimismo, a pesar de que su formulación en elementos finitos deba ser necesariamente distinta, los anteriores objetos también pueden ser combinados con `campp.1`, que representa a un campo de fuerzas actuante sobre un pilar. Esto es posible gracias a que los objetos–carga resultan independientes y autónomos respecto de sus objetos precursores tales como los soportes geométricos o las formulaciones mecánicas. De esta manera se consiguen objetos–carga del mismo tipo, siendo así posible su combinación directa.

A su vez `coefr.1`, `coefr.2` y `coefp.1` representan los factores de escala (objetos de tipo número real) por los que deben multiplicarse (operador *) los campos `campr` definidos en los nodos de la estructura. El resultado final es el objeto `campforj` que representa la superposición a nivel de toda la estructura –gracias a la actuación polimórfica de +– del conjunto de campos de cargas.

Régimen elástico lineal, homogéneo e isótropo:

En la Figura 1 se muestra el campo de sobrecargas adoptado para el estudio de una estructura de edificación compuesta por dos plantas, de planta cuadrada de 3×3 pilares. Éste se corresponde con lo dispuesto por la normativa española de acciones NBE–AE–88, siendo la sobrecarga alternada sobre los diversos recuadros de 0.4 T/m^2 (en rojo en la figura). Asimismo se ha considerado una carga lineal balconera de 0.2 T/m (color amarillo), así como la sobrecarga inducida en los voladizos por la existencia de cargas en los recuadros adyacentes (color verde), procedimiento que se realiza de manera automática. En la Figura 2 puede apreciarse parte de la deformada de la estructura, inicialmente calculada en régimen lineal elástico e isótropo, contrastada con parte de su geometría inicial.

elásticos lineales, homogéneos, ortótropos etc. La Figura 6 muestra la deformación de un depósito pretensado de 5000 m^3 de capacidad, vacío y apoyado sobre un suelo con un módulo de elasticidad vertical bilinealmente creciente con la profundidad. El pretensado perimetral está constituido por seis anillos concéntricos con una tensión después de pérdidas de 28.8 T cada uno.

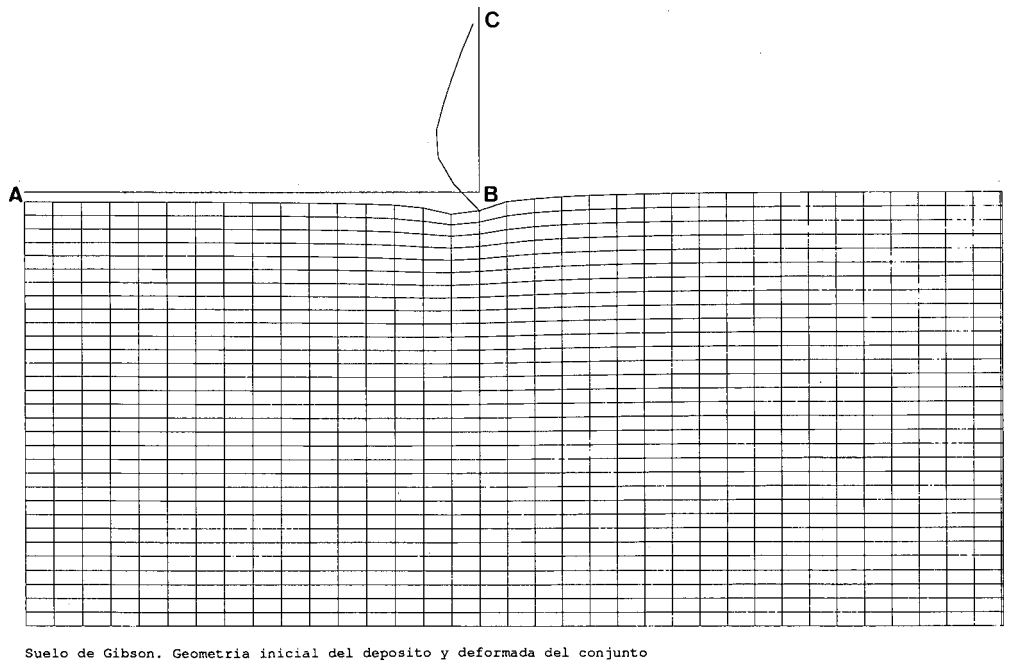


Figura 6. Depósito de hormigón armado pretensado exteriormente sobre suelo de Gibson. Configuración deformada (ampliada 1000 veces)

Problemas termomecánicos en elasticidad plana

Las posibilidades de refinamiento de un programa concreto no sólo pasan por el intercambio de unos objetos por otros. También es posible su combinación para analizar un nuevo problema. Así en la Figura 7a se muestra el estado tensional de una viga de gran canto, calculada según las hipótesis de la elasticidad plana, sometida a diversas acciones mecánicas. Al igual que en casos precedentes, la sustitución, tanto del modelo mecánico, como de la naturaleza de los campos de cargas –preservando el resto de objetos– posibilita el estudio de mismo sólido, esta vez bajo el efecto de un gradiente térmico (Figura 7b).

Sin embargo, si combinamos ambos estados de carga –incluyendo los dos modelos de respuesta, el mecánico y el térmico– estamos en condiciones de analizar la estructura bajo la actuación conjunta –parcialmente acoplada– de ambas acciones. Además, si modificamos el comportamiento inicial –lineal y elástico– del material de la viga por otro que contemple su eventual plasticidad, podemos obtener a través de un proceso incremental el punto de colapso del sólido. En la Figura 8 pueden apreciarse las tensiones de Von Mises asociadas a este problema termomecánico en el caso de la

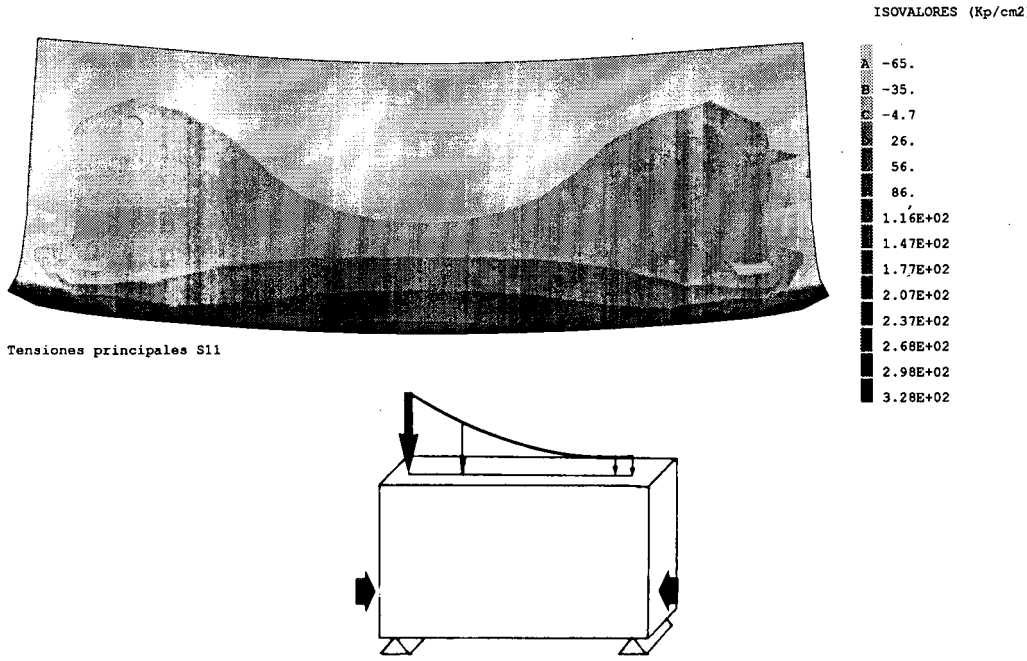


Figura 7a. Viga pared sometida a acciones mecánicas. Mapa de tensiones principales σ_{11} (Kp/cm^2)

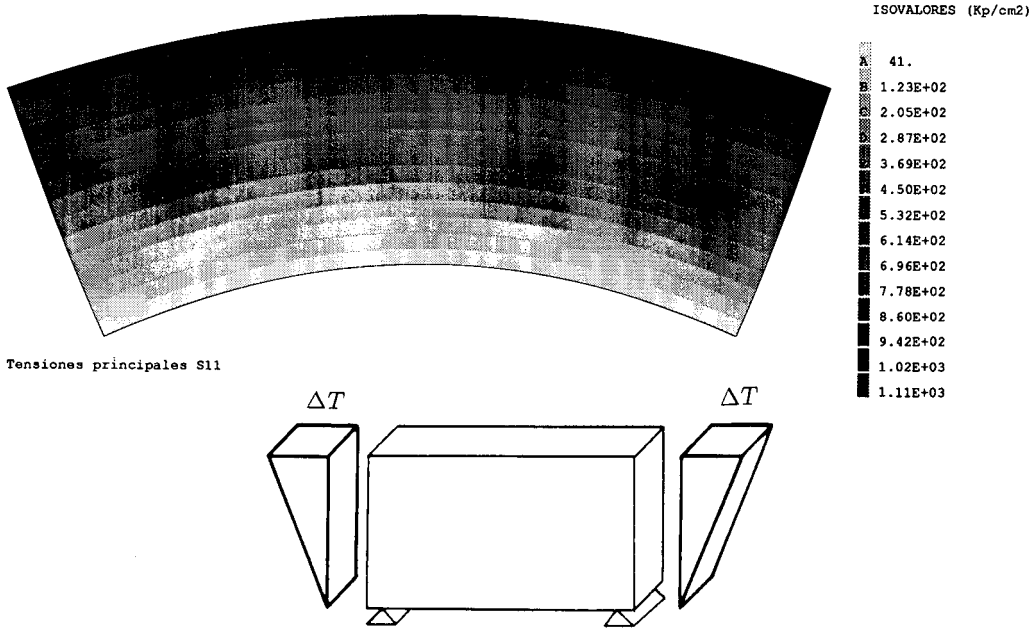


Figura 7b. Viga pared sometida a acciones térmicas. Mapa de tensiones principales σ_{11} (Kp/cm^2)

actuación del 10 %, 50 % y 90 % respectivamente de total de cargas de los dos casos anteriores.

El campo de tensiones de la Figura 7a resulta de interpretación un tanto oscura, por cuanto representa las isosuperficies de tensiones principales. Así resultaría muy útil contar con un metaoperador que transformara dicha imagen en un campo vectorial de tensiones orientadas definido sobre los nodos (Figura 9). A partir de este momento, esta nueva posibilidad de postproceso, generada en base a las posibilidades de representación preexistentes en el código, queda incorporada al conjunto de operadores disponibles.

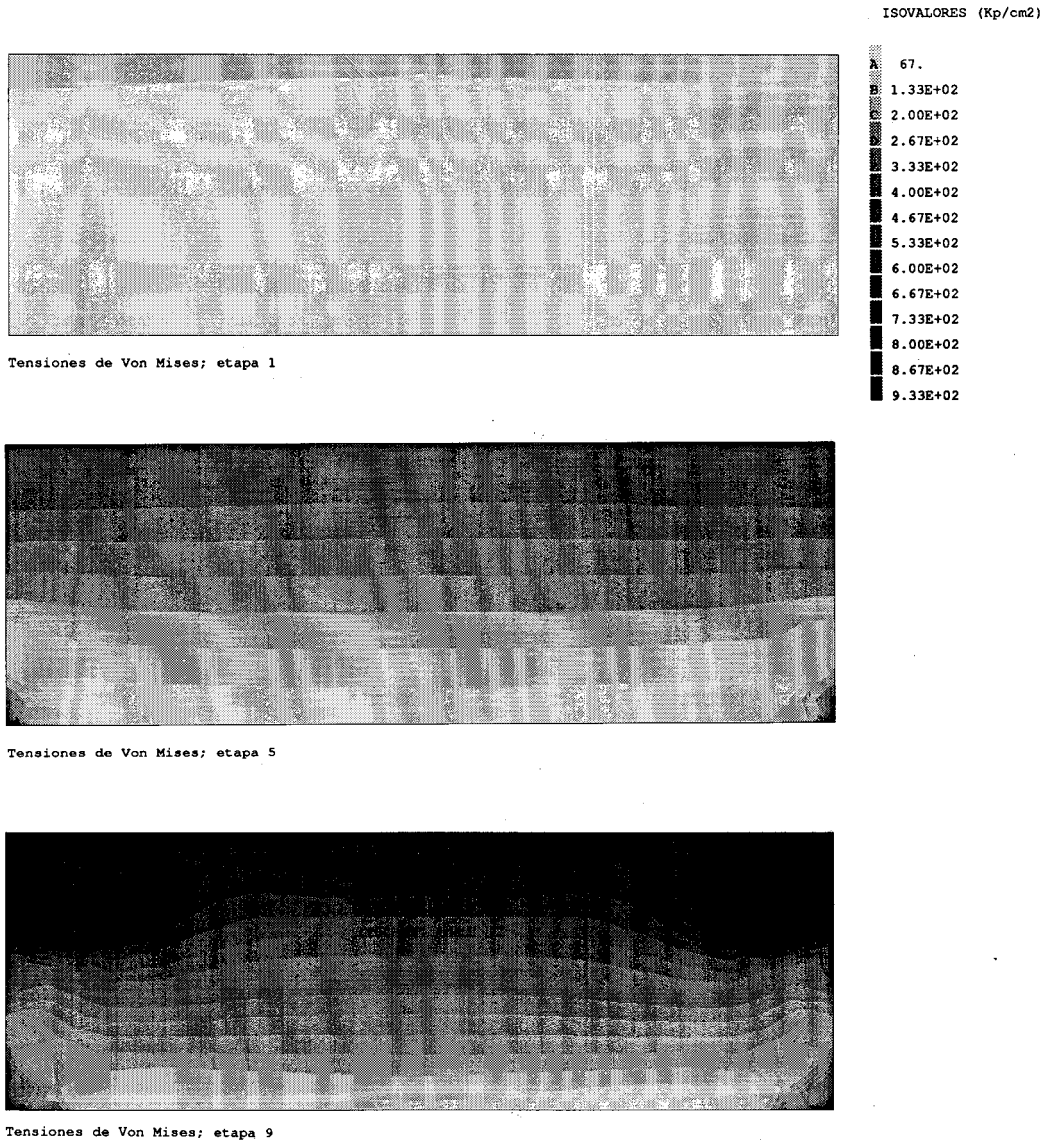


Figura 8. Viga pared sometida a acciones térmicas y mecánicas. Punto de reblandecimiento fijado de manera arbitraria en 900 Kp/cm^2 en tensión equivalente de Von Mises

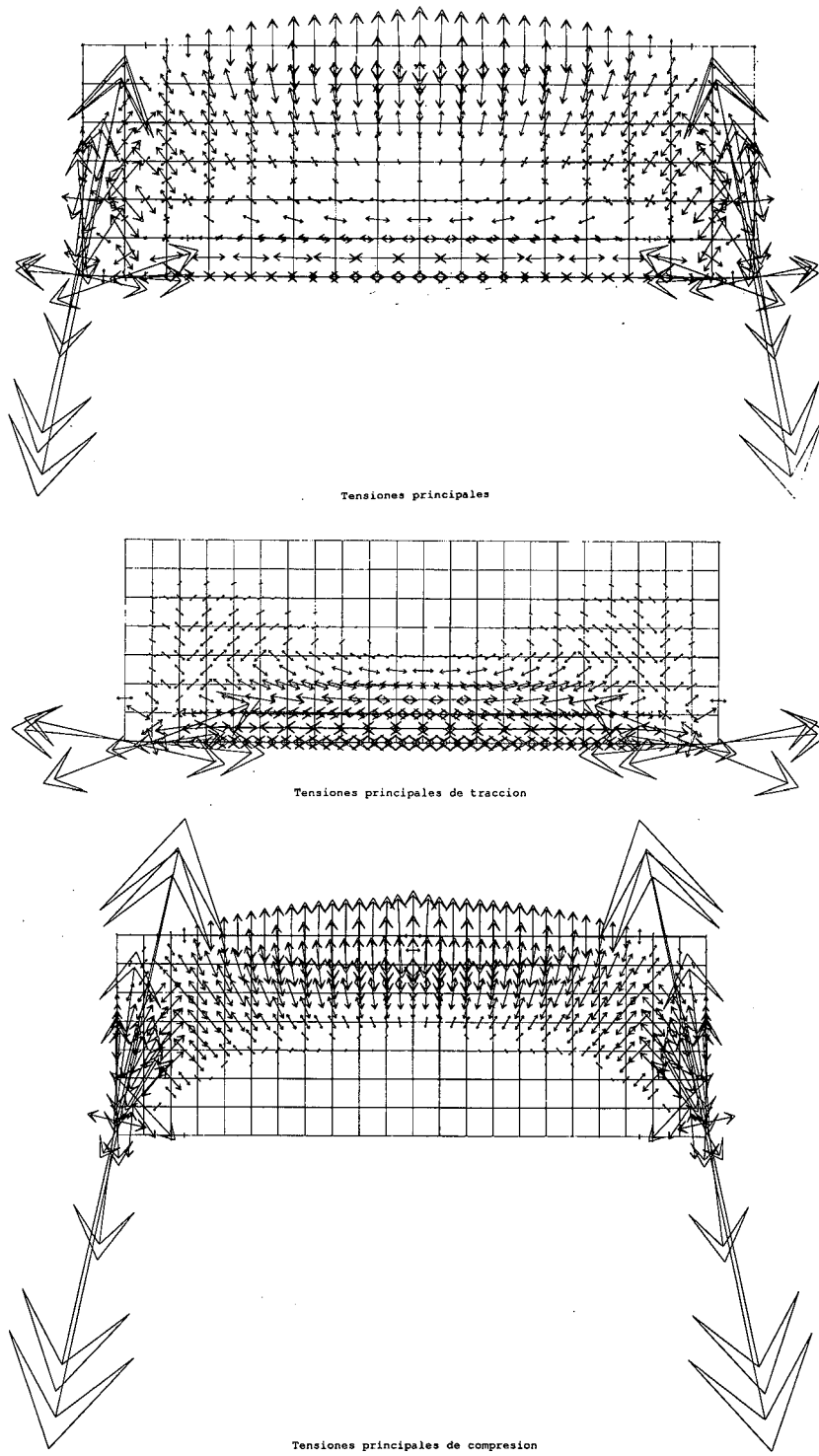


Figura 9. Viga pared sometida a acciones mecánicas. Actuación del metaoperador de postproceso; tensiones principales orientadas totales, de tracción y de compresión

Estructuras metálicas en régimen anelástico

La posibilidad de realizar cálculos no lineales, aunque ya ha sido tratada parcialmente en ejemplos precedentes, queda nuevamente reflejada en el análisis de un pórtico metálico formado por perfiles comerciales de acero laminado, sometido a la acción de diversas cargas estáticas. La ley constitutiva del material corresponde a lo dispuesto por la normativa española NBE–MV–103.

La Figura 10 muestra las sucesivas deformadas de la estructura a lo largo del proceso incremental de carga; los trazos en color verde corresponden a etapas de deformación elástica, siendo de color amarillo aquéllas en las cuales se ha alcanzado el régimen plástico en alguna zona de la estructura. En concreto, puede apreciarse la pérdida de ortogonalidad de la unión –inicialmente supuesta como empotrada– entre el dintel y los pilares, fruto de la aparición de rótulas plásticas. Paralemente, la Figura 11 refleja la evolución de los momentos flectores en el dintel.

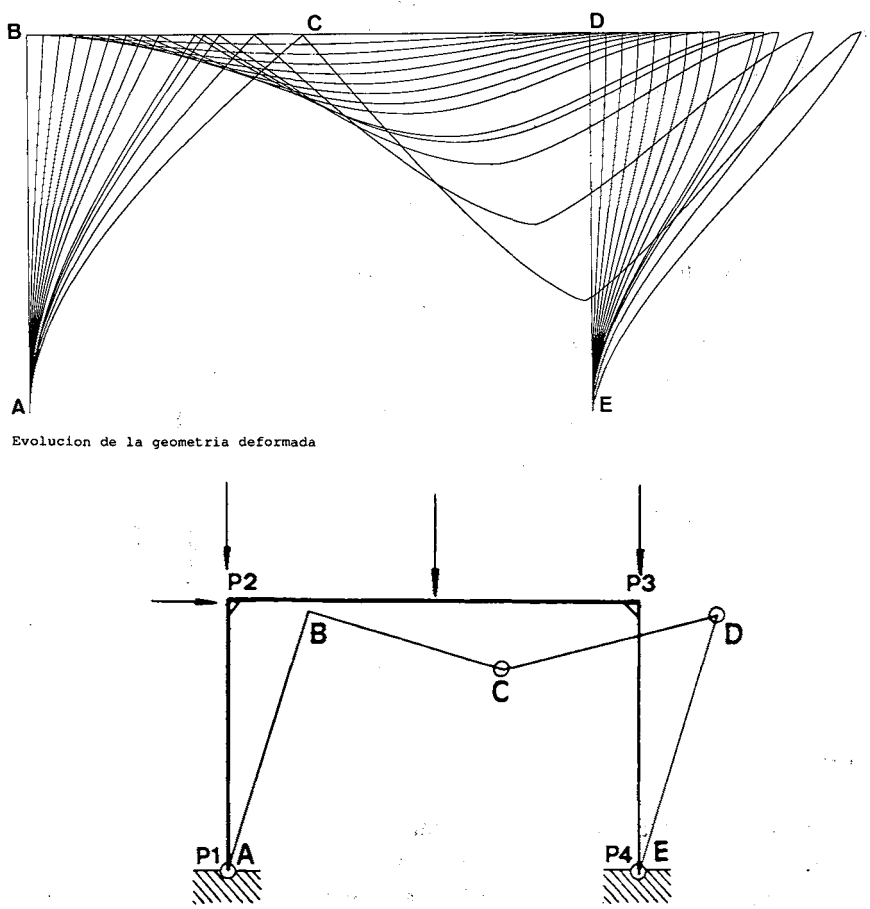


Figura 10. Pórtico metálico formado por dos pilares HEB140 y un dintel IPN180 en acero A37b. Estado de cargas, mecánismo de colapso y proceso de deformación (ampliado 1000 veces)

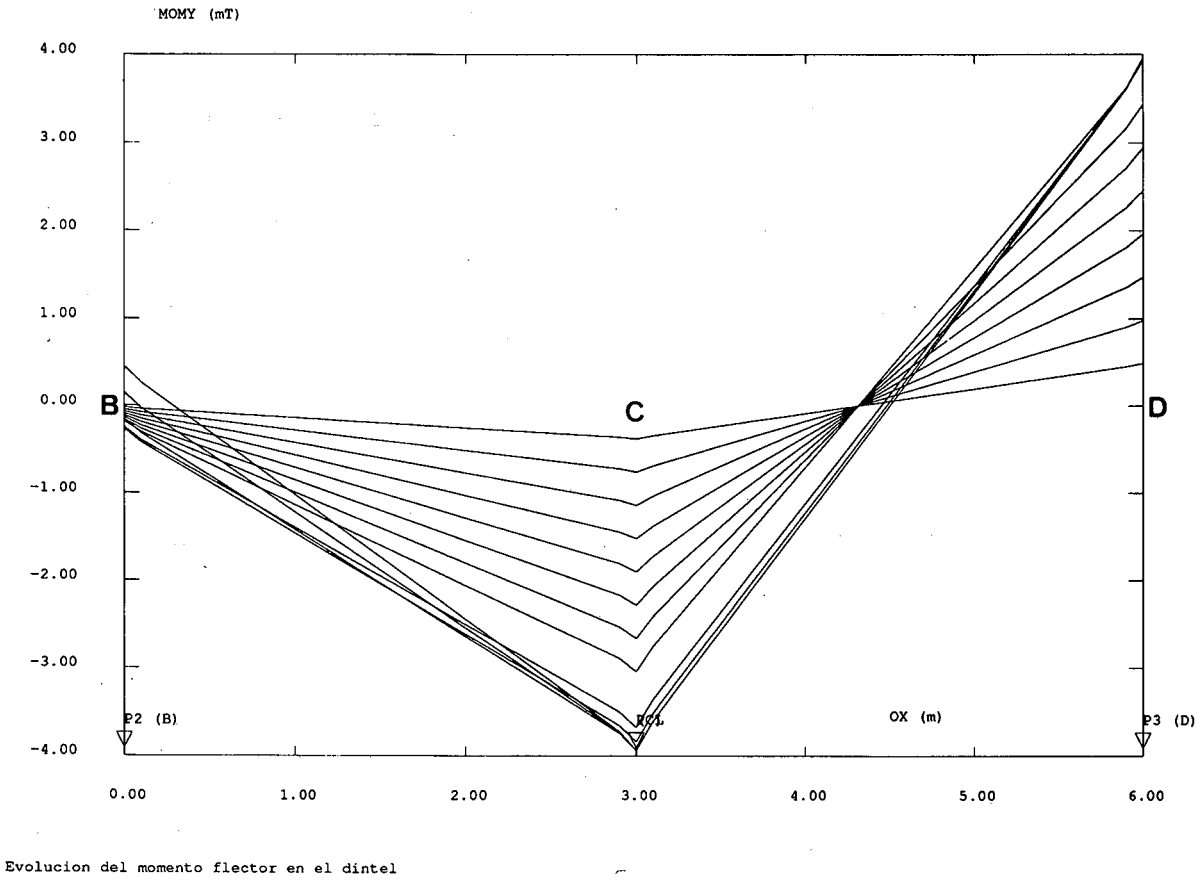
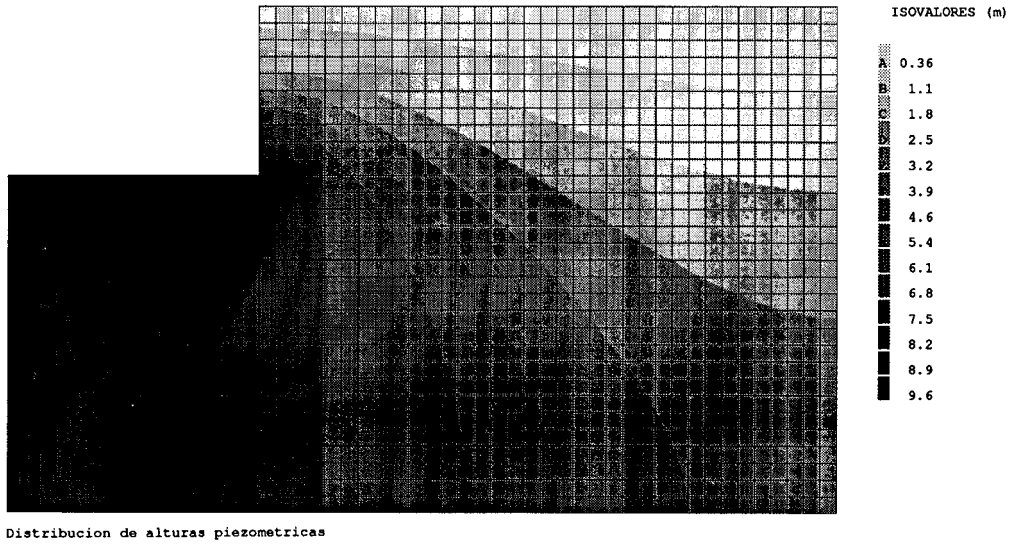


Figura 11. Pórtico metálico en régimen elasto–plástico perfecto. Momentos flectores en el dintel; rótulas plásticas en centro luz y en la unión con el pilar izquierdo

Filtración en medio poroso

La vocación de generalidad de los objetos –y eventualmente de los operadores– presentes en un COO facilita abordar problemas y situaciones inicialmente no previstas sin tener que recurrir a la implementación de nuevas clases o procedimientos. Así en la Figura 12 se muestra la distribución de alturas piezométricas para un problema de filtración en un medio homogéneo y ortótropo. La sección de la figura trata de idealizar el fenómeno –en régimen estacionario– de flujo en el fondo de una excavación. Inicialmente, se considera la totalidad del estrato saturado.

Gracias a la analogía matemática existente entre las ecuaciones que gobiernan este problema y el caso térmico, se pueden utilizar los objetos y operadores de este tipo presentes en el código para su resolución, debiendo cuidar únicamente la reinterpretación correcta de los resultados y magnitudes resultantes del cálculo. Asimismo y como ya sucedía en casos anteriores, resulta inmediato el refinamiento del modelo inicial para estudiar, para el mismo problema, el efecto de una tablestaca metálica hincada en la pared de la excavación (Figura 13). Para ello se sustituye el objeto "malla de elementos finitos" por uno nuevo que contemple la nueva geometría de nuestro problema. De la misma manera también puede evaluarse el caudal filtrado bajo la excavación, pudiendo obtener así la clava óptima de la pantalla.



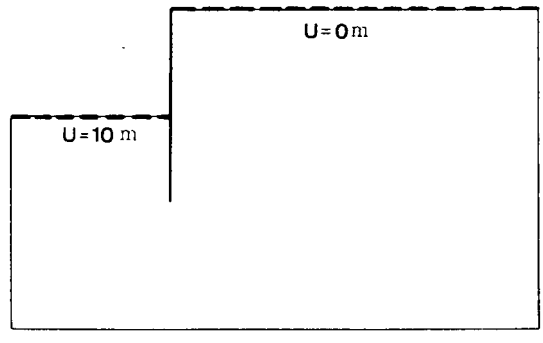


Figura 12. Filtración en medio poroso. Coeficiente de permeabilidad de 1 m/h en sentido horizontal. Condiciones de contorno y ley de alturas piezométricas (m), régimen estacionario

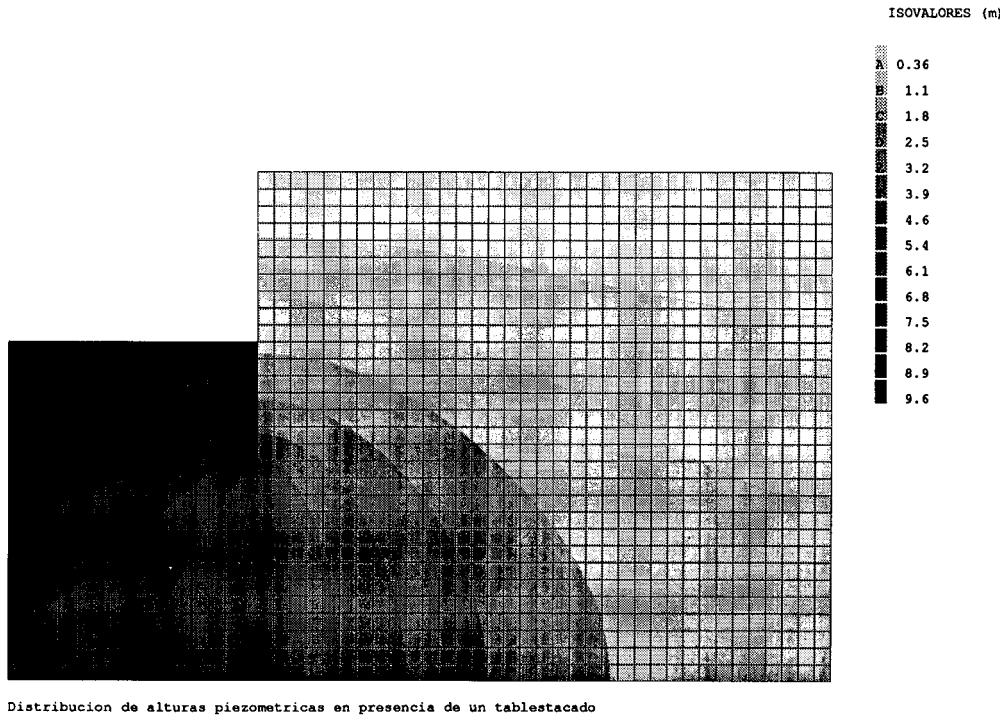


Figura 13. Filtración en medio poroso. Tablestaca hincada de clava igual a 4/5 de la altura de excavación. La reducción en el caudal bajo solera es del 47 %, respecto al caso de la Figura 12

Otras posibilidades de desarrollo del código

Finalmente, la ortogonalidad –entendida como independencia– de los diversos objetos manejados por un COO facilita una nueva vía de desarrollo no contemplada anteriormente. Hasta ahora, cuando un determinado objeto o clase no podía ser derivado por herencia –total o parcial– de lo ya existente, debíamos recurrir a la modificación de las fuentes del programa, al igual que sucedía con los lenguajes convencionales. Sin embargo, los procedimientos de generación de determinados objetos pueden ser de naturaleza suficientemente compleja como para constituir por ellos mismos auténticos programas, incrementándose notablemente la dificultad de su integración en el código.

Así y desde el punto de vista de CASTEM2000, que incorpora el preproceso como parte de sus facilidades, una alternativa a la implementación de nuevos generadores de malla diferentes al ya existente, consistiría en hacer compatible ambos programas. En efecto, conociendo el sistema de almacenamiento de la información propia de nuestro COO y dotando de la estructura de información característica de un objeto al resultado de otro preprocesador –que puede ser perfectamente un programa codificado en FORTRAN, orientado a la función– posibilitaremos que el código lo acepte como

propio, esto es, sin ser esencialmente diferente de cualquier otro objeto derivado por herencia.

Para llevar a cabo satisfactoriamente este proceso es fundamental incluir en ese nuevo objeto toda la información que separa el resultado de cualquier programa convencional de lo que hemos venido definiendo como tal. Por otra parte, si los objetos no incluyen como información de base sus posibilidades de modificación, se reducirá notablemente esta cantidad de información añadida, que es donde radica gran parte de la dificultad del método.

En la Figura 14 puede contemplarse un objeto generado según el preprocesador propio de CASTEM2000, mientras que el otro objeto representado, correspondiente a una malla alternativa definida sobre la misma geometría, ha sido obtenido por un preprocesador tradicional e implantado en el código a través de un interfase escrito también en FORTRAN. Para este caso patológico resulta ser de mayor calidad la malla formada únicamente por cuadriláteros, que ahora puede ser adoptada para posteriores análisis en el entorno del COO.

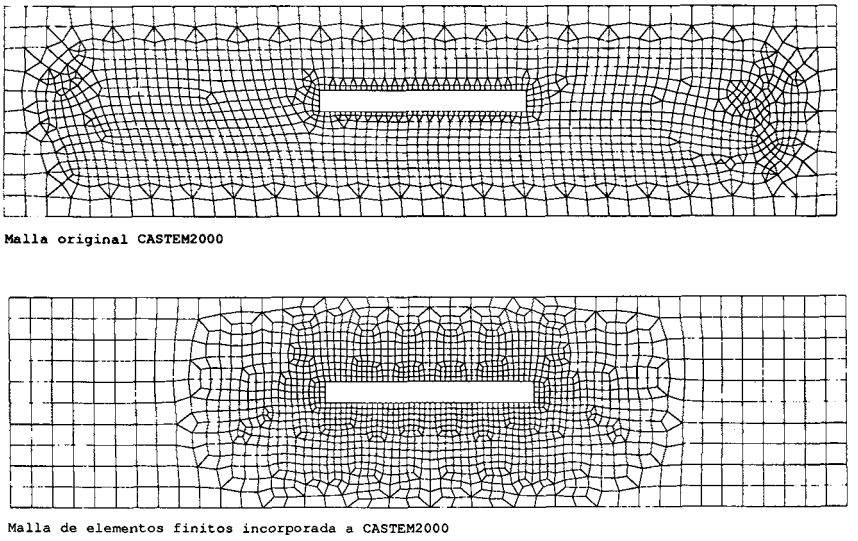


Figura 14. Mallas de elementos finitos. Arriba la que proporciona el COO y abajo malla alternativa generada independientemente e implantada con un *driver*

CONCLUSIONES

A lo largo de los anteriores apartados se ha tratado de caracterizar la programación orientada al objeto, las ideas básicas en que se sustenta así como los elementos que la integran. La idea básica es almacenar la información en objetos de estructura compleja que puedan manejarse a través de procedimientos sencillos de ejecución pero de interpretación abstracta. La POO no es un fin en sí misma, sino un medio para

efectuar, en nuestro caso, cálculos por el MEF. Es desde ese punto de vista que debe ser considerada como herramienta bajo la doble perspectiva de su utilización presente –con los recursos actualmente disponibles para cada código en concreto– y de futuros desarrollos.

Estos códigos se presentan como una alternativa muy válida a los programas convencionales, en tareas de cálculo rutinario pero sobretodo a nivel de implementación de nuevas posibilidades. Entre sus principales ventajas pueden citarse la facilidad de manejo de los diversos objetos, la accesibilidad permanente a toda la información que éstos contienen, el posible carácter interactivo del lenguaje, la posibilidad de compartir los desarrollos de diversos grupos de trabajo, su natural facilidad para efectuar refinamientos progresivos de los programas y su adaptabilidad a la resolución de problemas o situaciones inicialmente no previstas.

Mediante algunos ejemplos se ha mostrado la posibilidad de emplear un código orientado al objeto en la resolución de problemas de ingeniería. Cada caso concreto mostrado responde a una combinación diferente de los diversos operadores y objetos necesarios para su resolución y disponibles en nuestro COO en particular. A su vez se evidencia la facilidad para dar respuesta a cada uno de los problemas planteados, una vez conocidos los recursos a emplear; en concreto, la mayoría de estos ejemplos apenas supera las 50 instrucciones para su total resolución, postproceso incluido.

No podemos, por otra parte, dejar de remarcar las dificultades derivadas de este nuevo concepto de programación, cuales son el recurso a medios informáticos de grandes prestaciones, o el frecuente desconocimiento del funcionamiento y características internas de objetos y operadores, que puede conducir a equívocos derivados de su utilización incorrecta pero inconsciente.

Asimismo el potencial usuario de un código de estas características debe conocer los fundamentos del MEF, porque sólo así se puede esperar su correcta utilización; muchas de las combinaciones de operadores y objetos pueden tener formalmente sentido –es decir, no provocar un error de programa– pero resultar incoherentes o inapropiadas a la hora de resolver un problema concreto. Estas situaciones siempre conducen a soluciones erróneas, razón por la cual es todavía más importante que en el caso de los programas convencionales el análisis crítico de los resultados.

Por todo ello resulta fundamental disponer de una completa y actualizada documentación del código a la hora de iniciar su utilización, pese a lo atractivo que resulta comenzar a obtener desde el primer instante resultados, en base a la facilidad con que pueden ser resueltos problemas de dificultad media. Es precisamente en su gran inercia de uso donde radica el mayor peligro.

Con todo y a pesar de estos inconvenientes, por otra parte soslayables con relativa facilidad, la POO constituirá en un futuro –si no lo hace ya– un punto de referencia obligado a la hora de hablar de la nueva generación de códigos de cálculo.

REFERENCIAS

1. P. Verpeaux et A. Millard, "De l'existence à l'essence. De CASTEM á CASTEM2000. Quelques considérations sur le développement de grand codes du calcul", *Rapport 88/179, Laboratoire d'Analyse Mécanique des Structures*, C.E.A., (1988).

2. Y. Dubois-Pélerin, T. Zimmermann and P. Bomme, «Object-Oriented Finite Element Programming Concepts», *New Advances in Computational Structural Mechanics,* pp. 457-462, Elsevier, (1992).
3. G.R. Miller, «An Object-Oriented Approach to Structural Analysis and Design», *Computers & Structures, Vol. 40, No. 1, pp. 75-82, (1991).*
4. D. Lucas, B. Dressler and D. Aubry, «Object-Oriented Finite Element Programming Using ADA Language», *Proceedings of the First European Conference on Numerical Methods in Engineering,* Elsevier, pp. 591-598, (1992).
5. M. Galindo, «FemLab version 1.0. Technical Report No. IT-114», CIMNE, International Center for Numerical Methods in Engineering, Barcelona, (1994).
6. B. Stroustrup, «The C++ Programming Language», Reading, MA, Adison-Wesley, (1986).
7. P. Verpaux, «Regles de programmations dans CASTEM2000», *Rapport 91/254, Laboratoire d'Analyse Mécanique des Structures,* C.E.A., (1991).
8. P. Pegon, «Model Implementation in CSTEM2000: Some General Considerations and a Simple Practical Realization», *Technical Note,* No. I.93.06, Safety Technology Institute, Commission of the Communities, Joint Research Center, (1993).
9. S. Ortiz, «Simulació numérica d'ones sòniques», Tesina de especialidad, ETSECCPB, (1993).
10. M.A. Bretones, A. Rodríguez Ferran y A. Huerta, «Programación orientada al objeto, una herramienta de ingeniería y desarrollo», *Proceedings del segundo congreso de métodos numéricos en ingeniería,* SEMNI, pp. 273-282, (1993).
11. M.A. Bretones, «Programación orientada al objeto, una herramienta de ingeniería y desarrollo», Vols. 1 y 2, Tesina de especialidad ETSECCPB, (1993).