novática
UPGRADE Revista de la Asociación de Técnicos de Informática

Novática, revista fundada en 1975 y decana de la prensa informática española, es el órgano oficial de expresión y formación continua de **ATI** (Asociación de Técnicos de Informática). ***Novática*** edita también ***Upgrade***, revista digital de **CEPIS** (*Council of European Professional Informatics Societies*), en lengua inglesa, y es miembro fundador de **UPENET** (*UPGRADE European NETwork*)

<http://www.ati.es/novatica/>
<http://www.upgrade-cepis.org/>

ATI es miembro fundador de **CEPIS** (*Council of European Professional Informatics Societies*) y tiene un acuerdo de colaboración con **ACM** (*association for Computing Machinery*). Tiene asimismo acuerdos de vinculación o colaboración con **AdaSpain**, **AI2** y **ASTIC**.

Coordinación Editorial, Redacción Central y Redacción ATI Madrid
Padilla 66, 3º, dcha., 28006 Madrid
Tlfn.914029391; fax.913093685 <novatica@ati.es>
Composición, Edición y Redacción ATI Valencia
Av. del Reino de Valencia 23, 46005 Valencia
Tlfn./fax 963330392 <secreval@ati.es>
Administración y Redacción ATI Cataluña
Via Laietana 41, 1º, 1ª, 08003 Barcelona
Tlfn.934125235; fax 934127713 <secregen@ati.es>
Redacción ATI Andalucía
Isaac Newton, s/n, Ed. Sadiel,
Isla Cartuja 41092 Sevilla, Tlfn./fax 954460779 <secreand@ati.es>
Redacción ATI Aragón
Lagasca 9, 3-B, 50006 Zaragoza.
Tlfn./fax 976235181 <secreara@ati.es>
Redacción ATI Asturias-Cantabria <gp-astucant@ati.es>
Redacción ATI Castilla-La Mancha <gp-clmancha@ati.es>
Redacción ATI Galicia
Recinto Ferial s/n, 36540 Silleda (Pontevedra)
Tlfn.986581413; fax 986580162 <secregal@ati.es>
Suscripción y Ventas
<http://www.ati.es/novatica/interes.html>, o en ATI Cataluña o ATI Madrid
Publicidad
Padilla 66, 3º, dcha., 28006 Madrid
Tlnf.914029391; fax.913093685 <novatica.publicidad@ati.es>
Imprenta
9 Impressió S.A., Juan de Austria 66, 08005 Barcelona.
Depósito Legal: B 15.154-1975 -- ISSN: 0211-2124; CODEN NOVAEC
Portada: Antonio Crespo Foix / © ATI 2004
Diseño: Fernando Agresta / © ATI 2004

Nº 169, mayo-junio 2004, año XXX

sumario

Monografía del próximo número: "Agentes Software"

Un refactorizador simple

José A. Leiva Izquierdo[1], Ángel Herranz Nieva[2]
[1] Plettac Seguridad y Sistemas; [2] Facultad de Informática, Universidad Politécnica de Madrid

<jleiva@ya.com>, <aherranz@fi.upm.es>

El enunciado de este problema apareció en el número 168 de **Novática** (marzo-abril 2004, p. 71). Es el problema D de los planteados en el I Concurso Universitario de la Comunidad Autónoma de Madrid (CUPCAM 2003)

1. Resumen del problema

El problema consistía en programar un *refactorizador* de código que básicamente moviera operaciones de un módulo a otro con el objetivo de disminuir el acoplamiento entre todos los módulos que constituyen el sistema. Es obvio que el problema es una abstracción trivial del concepto de acoplamiento y que se ignoran infinidad de dificultades reales. El acoplamiento total de sistema se mide mediante la suma de apariciones estáticas de llamadas entre operaciones de distintos módulos. Como información de entrada al problema se recibían módulos, métodos, pertenecía de métodos a módulos y llamadas de métodos a métodos. La salida que se espera es el valor del acoplamiento mínimo tras aplicar la refactorización.

2. Grafos y colores

Es posible aplicar alguna variante de coloración de grafos. Cada método es un nodo (de un color determinado por pertenecer a un módulo concreto). Una arista (dirigida) existe entre dos nodos si hay una llamada de un método a otro. El objetivo final es minimizar el número de aristas que parten de un nodo y llegan a otro de diferente color.

Mediante un algoritmo iterativo sería posible estabilizar el grafo llevando a cada método al módulo óptimo en su caso. En cada paso del algoritmo se seleccionaría un método cuyo acoplamiento fuera positivo, existiendo un módulo tal que al pertenecer se disminuyera el acoplamiento general del programa.

3. Algoritmo

Se propone una sencilla implementación en Java para modelar el dominio descrito y aplicar el sencillo algoritmo iterativo propuesto. El primer paso es modelar el dominio donde deben coexistir métodos, módulos (también llamadas clases) y el programa como tal. Se han omitido del código los métodos de acceso a atributos y aquellos encargados de analizar la entrada.

Todos los objetos del dominio implementan un método encargado de llevar a cabo la tarea de optimización. El verdadero paso de iteración se da en la clase encargada de definir un método. Los objetos módulo y programa gestionan el trabajo realizado por la clase método y se detienen en el momento en que un paso no ha implicado mejoría.

Bien es cierto que esta solución puede encontrarse el problema de detener la ejecución creyendo haber llegado a una solución óptima cuando en realidad no es así. Sin embargo en pro de simplificar la solución el conjunto de datos de partida fue reducido voluntariamente asumiendo el grado de error.

```
class Method
{
  private int id;
  private Module module;
  private Vector called;

  public Method(int id, Module module)
  {
    called = new Vector();
    this.setModule(module);
    this.setId(id);
  }

  public boolean optimise()
  {
    boolean result = false;
    int highest =
       getMostFrecuentCalledModule();
    if (highest != module.getId()) {
      module.getProgram().moveMethod(
        module.getId(),
        getId(),
        highest);
      result = true;
    }
    return result;
  }

  private int
    getMostFrecuentCalledModule()
  {
    int highest = this.module.getId();
    int[] sources =
      new int[module.getProgram().
                getModulesAmount()];

    for (int i = 0;
         i < called.size();
         i++)
      sources[
        ((Method)called.elementAt(i)).
          getModule().getId()
      ]++;

    for (int i = 0;
         i < sources.length;
         i++)
      if (sources[i]
          > sources[highest])
        highest = i;
    return highest;
  }

  public int getForeignCalls()
  {
    int result = 0;

    for (int i = 0;
         i < called.size();
         i++)
     if (((Method)called.
           elementAt(i)).
             getModule().getId()
         != module.getId())
       result++;
    return result;
}
```

El objeto módulo, definido como clase en la demostración posee un conjunto de métodos sobre los que aplica el algoritmo de optimización.

```
class Module
{
  private int id;
  private Vector methods;
  private Program program;
  private int nextMethodId;

  public boolean optimise()
  {
    boolean improved = false;
    for (int i = 0;
         i < methods.size();
```

```
        i++)
      improved =
        improved ||
        ((Method)methods.
           elementAt(i)).
             optimise();
    return improved;
  }

  public void addMethod(Method method)
  {
    method.setModule(this);
    method.setId(nextMethodId);
    methods.add(method);
    nextMethodId++;
  }

  public void delMethod(Method method)
  {
    methods.remove(method);
  }
  public void reIndexMethods()
  {
    for (int i = 0;
         i < methods.size();
         i++)
      ((Method)methods.elementAt(i)).
        setId(i);
  }

  public int getForeignCalls()
  {
    int result = 0;

    for (int i = 0;
         i < methods.size();
         i++)
      result +=
        ((Method)methods.
           elementAt(i)).
             getForeignCalls();
    return result;
  }
```

Un programa está compuesto por un conjunto de módulos. Se muestran el método que desencadena el proceso de optimización y el implicado en la aplicación del patrón implicado en el desplazamiento de código de un módulo a otro.

```
public class Program
{
  private Vector modules;

  public Program()
  {
    modules = new Vector();
  }

  public void optimise()
  {
    boolean improved = true;
    while (improved) {
      improved = false;
      for (int i = 0;
           i < modules.size();
           i++)
        improved =
          improved ||
          ((Module)modules.
             elementAt(i)).
               optimise();
    }
    reIndexMethods();
  }

  private void reIndexMethods()
  {
    for (int i = 0;
         i < modules.size();
         i++)
      ((Module)modules.elementAt(i)).
        reIndexMethods();
  }
  public void addModule(Module module)
  {
    modules.add(module.getId(),
                module);
  }

  public void
    moveMethod(int fromModule,
               int idMethod,
               int toModule)
  {
    Method method =
       getModule(fromModule).
         getMethod(idMethod);
    getModule(fromModule).
      delMethod(method);
    getModule(toModule).
      addMethod(method);
  }

  public int getForeignCalls()
  {
    int result = 0;
    for (int i = 0;
         i < modules.size();
         i++)
      result +=
        ((Module)modules.
           elementAt(i)).
             getForeignCalls();
    return result;
  }
```

Por último la clase `Refactorer` es la encargada de tomar la entrada estándar, construir el dominio y ejecutar el algoritmo de optimización.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Refactorer {

  private Program program;

  public void ioRun()
  {
    try {
      BufferedReader br =
        new BufferedReader(
          new InputStreamReader(
                System.in));
      program = new Program();
      while (!program.parse(br)) {
        program.optimise();
        System.out.println(
           program.getForeignCalls());
      }
    } catch (IOException ioe) {
      System.out.println(
        "IOException: "
        + ioe.getLocalizedMessage());
    }
  }
}
```

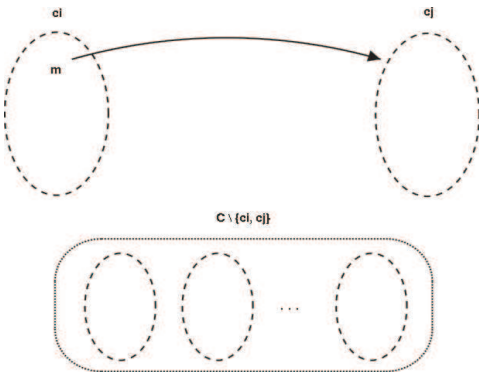

Figura 1. Prueba.

4. Terminación

Nos asalta una duda: ¿el algoritmo propuesto termina? Para ello nuestro objetivo será demostrar como el acoplamiento general del programa disminuye en cada iteración. Se muestran unas definiciones previas para indicar a continuación la condición que se debe cumplir para desplazar un método entre módulos. Dado un programa, utilizaremos la siguiente notación:

Conjunto de clases $C=c_0,c_1,c_2,\ldots\ c_i$

Conjunto de métodos $M=m_0,m_1,m_2,\ldots$ mi tal que $\forall m\in M.\ \exists c\in C.m\in c$

Grado de salida de un método $g_s(m_i)$ Número de llamadas a procedimientos que realiza

Grado de entrada de un método $g_e(m_i)$ Número de métodos que le llaman

Grado de salida a una clase $g_s(m_i,c_j)$ Número de llamadas a procedimientos de la clase c_j

Grado de entrada de una clase $g_e(m_i,c_j)$ Número de procedimientos de la clase c_j que llaman

Acoplamiento de un método Diferencia entre las llamadas a métodos de otras clases con respecto a métodos de su propia clase. Sea el método m_i y la clase c_j con $m_i<c_j$ se define el acoplamiento de m_i como

Acoplamiento de programa

$$A_G = \sum_{C}^{m_i} A(m_i)$$

Cuando un método se desplaza de clase tal y como muestra la **figura 1** el acoplamiento de los métodos de la clase c_i que llamaban a m aumenta mientras que el acoplamiento de m y el de los métodos de c_j que llaman a m disminuyen.

Dado que el objetivo es lograr que el acoplamiento disminuya en cada paso del algoritmo es necesario asegurarse de que el aumento que provocan los métodos de c_i sea inferior a la disminución lograda con m y los métodos de c_j. Las condiciones a tener en cuenta serían las siguientes:

$$g_e(m,c_i) < g_e(m,c_j) + g_s(m,c_j) \quad (1)$$
$$g_s(m,c_j) < g_s(m,c_j) \quad (2)$$

También se podrían sustituir las dos condiciones por una sola:

$$g_e(m,c_i) + g_s(m,m_i) < 2(g_e(m,c_j) + g_s(m,c_j)) \quad (3)$$

coordinación editorial

Novática disponible en la Intranet de ATI

Recordamos que en la Intranet de ATI, en <http://intranet.ati.es/novatica/>, se encuentran disponibles, en formato PDF, las versiones digitales completas de todos los números de nuestra revista a partir del 138 (feb.-mar. 1999), además de los números 0 y 1, incorporados recientemente.

El acceso está restringido a los socios de ATI únicamente, con la única condición de que estén registrados en ATInet. Para registrase hay que ponerse en contacto con <info@ati.es>.

Noticias de *UPGRADE* y *UPENET* (*UPGRADE European NETwork*)

En el número de junio 2004 de ***UPGRADE***, <http://www.upgrade-cepis.org>, se ha incorporado a ***UPENET*** (*UPGRADE European NETwork*) la revista digital ***Mondo Digitale***, <http://www.monododigitale.net>, editada por AICA (*Associazione Italiana per l'Informatica ed il Calcolo Automatico* . En dicho número aparece un artículo de esta revista italiana, seleccionado por su director, ***Franco Filippazzi***.

Tras esta nueva incorporación, ***UPENET***, la recientemente creada red de publicaciones de sociedades miembro de CEPIS (*Council of Professional Informatics Societies*, <http://www.upgrade-cepis.org>), cuenta ya con tres miembros: ***Novática***, ***Mondo Digitale*** y ***Pro Dialog***, revista polaca coeditada por PTI-PIPS (*Polskie Towarzystwo Informatyczne – Polish Information Processing Society*) y por el Instituto de Informática de la Universidad Técnica de Poznan.

Tras el verano está prevista la incorporación de revistas de otras sociedades miembro de CEPIS, de lo cual podrán beneficiarse los lectores de ***Novática***, pues las revistas que forman parte de esta red, entre ellas la nuestra, podrán a su vez publicar artículos aparecidos en ***UPGRADE*** y en las revistas participantes en la misma ... y también se beneficiarán los autores pues en ***UPGRADE*** podrán publicarse artículos seleccionados de las revistas que forman parte de ***UPENET***.

Recordamos que ***UPGRADE*** tiene una lista cuyo objetivo es distribuir noticias sobre esta revista digital. En <http://www.upgrade-cepis.org/pages/editinfo.html#newslist> se encuentra información sobre sus características y sobre cómo darse de alta en élla.

programación de novática

Monografías para 2004

Por acuerdo de los Consejos Editoriales de ***Novática*** y ***Upgrade*** los temas de las restantes monografías del año 2004, salvo imprevistos o causas de fuerza mayor, serán los siguientes:

Nº 170 (julio-agosto): "*Agentes Software* ".
Editores invitados: ***Pedro Cuesta Morales***, <pcuesta@uvigo.es> y ***Juan Carlos González***, <jcmoreno@ei.uvigo.es> (Universidad de Vigo), ***Zahia Guessoum*** (Université Paris VI, Francia), <Zahia.Guessoum@lip6.fr> y ***Juan Pavón Mestras*** (Universidad Complutense de Madrid), <jpavon@sip.ucm.es>.

Nº 171 (septiembre-octubre): "*Tecnologías de Proceso Software* "
Editores invitados: ***Gerardo Cánfora*** (Università di Sannio, Italia), <gerardo.canfora@unisannio.it>, y ***Francisco Ruiz González*** (Universidad de Castilla - La Mancha), <francisco.ruizg@uclm.es>.

Nº 172 (noviembre-diciembre): "*Criptografía* ".
Editores invitados: ***Javier Areitio Bertolín*** (Universidad de Deusto), <jareitio@orion.deusto.es>, ***Arturo Ribagorda Garnacho*** (Universidad Carlos III de Madrid), <arturo@inf.uc3m.es>.