Refactorización en Código Fortran Heredado

Mariano Méndez[1], Alejandra Garrido[1,*], Jeffrey Overbey[2], Fernando G. Tinetti[1,**], and Ralph Johnson[2]

[1] Fac. de Informática, Universidad Nacional de La Plata,
50 y 120, La Plata, Buenos Aires, Argentina
[2] Dept. of Computer Science, University of Illinois at Urbana-Champaign,
201 N. Goodwin Ave., Urbana, Illinois, USA

Resumen Este artículo exhibe los obstáculos hallados por los programadores en las tareas de mantenimiento de software heredado escrito en Fortran. Por un lado se señalan las dificultades que se presentan en los procesos de comprensión, adaptación y mejora de este tipo de software. Por otro lado se encuadra una solución a este problema proponiendo la refactorización como técnica para ser aplicada en dichos procesos, intentando además, desmitificar los prejuicios que Fortran ha adquirido dentro del ámbito de la producción de software. Asimismo se describe detalladamente la implementación de algunas refactorizaciones en Photran, una nueva herramienta gráfica automatizada de refactorización para Fortran.

Key words: Refactoring. Fortran. Software Heredado

1. Introdución

Uno de los problemas más serios a los que Fortran se ve sometido actualmente como lenguaje de programación tiene su origen en su propia historia. Nacido hacia fines de 1956 [5], "The IBM Mathematical Formula Translation System" es el primer lenguaje de programación de alto nivel. El mismo ha tenido gran relevancia en el ámbito científico y de procesamiento paralelo, y se constituyó como pieza central en el desarrollo de software para la industria aeroespacial desde sus comienzos [30](ej: Misiones Mariner I y II). En su historia ha pasado por numerosas revisiones e incluso este año se prevée la publicación de una nueva versión.

Hoy en día contamos con un gran volumen de líneas de código Fortran, que en su mayor parte conforman código heredado, es decir, difícil de entender, mantener y extender. El problema se agrava porque además de la inherente complejidad del software [9], generalmente se trata de aplicaciones construidas hace mucho años a través de los cuales ha ido cambiando el equipo de desarrollo y mantenimiento.

En este artículo se plantean algunos de los problemas de mantener y actualizar código heredado en Fortran, y se propone la técnica de refactorización

[*] also LIFIA and CONICET Argentina
[**] also LIDI and Comisión de Investigaciones Científicas de la Prov. de Bs. As.

para solucionar eficientemente y automáticamente estos problemas, sin introducir errores y preservando el comportamiento del software.

La estructura de este artículo es la siguiente: en la Sección 2 se introducen los conceptos de Software Heredado y Refactorización. En la Sección 3 se realiza un análisis detallado de la evolución de Fortran a manera de motivar la necesidad de refactorización. En la Sección 4 se estudian las carácteristicas de refactorizaciones para software escrito en Fortran. En la Sección 5 se propone la implementación de la solución propuesta en Photran. La sección 6 presenta los trabajos relacionados y finalmente la Sección 7 muestra las conclusiones.

2. Refactorización y el Software Heredado

2.1. Código Heredado

Si bien existen controversias dentro de la definición de Código Heredado (Legacy Code), básicamente todos los autores que intentan definir este tipo de software coinciden con ciertas pautas dentro de sus definiciones [8,7,16]:

- es un tipo de software crítico para la organización.
- es muy complejo de cambiar, modificar o actualizar.

La definición utilizada en este artículo será la propuesta por Nicolas Gold [16] que resume al software heredado como: "Software crítico que no puede ser modificado eficientemente".

Cabe destacar que las modificaciones a realizar en dicho software varían en su naturaleza. Se entiende por modificaciones aquellos cambios destinados a: prevenir errores, corregir errores, introducir mejoras al software existente o agregar nueva funcionalidad. Para ello es importante contar con un buen entendimiento y comprensión del software y además poseer un adecuado conjunto de herramientas.

En este artículo se analizará el software heredado bajo la perspectiva de Fortran como lenguaje de programación debido a su particular característica evolutiva y a su trayectoria de más de 50 años que lo convierten en un lenguaje ideal para mostrar los problemas del código heredado y nuestra propuesta de solución.

2.2. La Refactorización

Software refactoring surge como "el proceso en el cual se aplican cambios en un sistema de software de forma tal que no altere el comportamiento externo del código, mejorando su estructura interna" [12]. Por definición, el proceso de refactorización de código fuente debe preservar el comportamiento del sistema, aunque no podemos encontrar en la bibliografía una definición consensuada de qué se entiende por comportamiento externo [22]. En este artículo adoptamos la definición original dada por Opdyke en su tesis, que sostiene que el conjunto de entrada y el conjunto de salida deben ser los mismos antes y después de aplicar el proceso de refactorización [23].

El primer uso conocido del término refactorización surgió en el contexto de la Programación Orientada a Objetos (POO), sobre el lenguaje C++ [23,24], como un intento de mejorar la producción de software reusable.

Posteriormente se extendió hacia otros paradigmas de programación, como por ejemplo la programación estructurada [15,26,20]. Fortran es un caso interesante para estudiar debido a que sus características evolutivas traen a la luz refactorizaciones específicas que otros lenguajes de programación no poseen, como ser refactorizaciones de actualización de sintaxis e incorporación de nuevas construcciones del lenguaje. La próxima sección describe brevemente la evolución del lenguaje Fortran, que motiva las refactorizaciones que proponemos [27,21].

3. Fortran a Través del Tiempo

La primera versión del lenguaje, llamada FORTRAN I, comprendía un total de 32 instrucciones. Posteriores versiones fueron aportando características requeridas según el avance de los lenguajes de programación. FORTRAN II, III y IV aportaron nuevos conceptos al lenguaje como el de subrutina, función y "common area" no incluidas en la versión original.

Con la especificación de FORTRAN 66 se logró alcanzar la primera publicación de un estándar ANSI para este lenguaje. En los años '70 se publicó una nueva versión del estándar, llamada FORTRAN 77 [3]. Ésta incluía algunas mejoras, principalmente el perfeccionamiento de las estructuras de control para la aplicación del paradigma de programación estructurada.

En la primera mitad de la década de los '90 otra revisión del estándar resultó en la publicación de Fortran 90, en la cual se introdujeron cambios como la supresión del formato fijo de FORTRAN I, se permitió que los identificadores superaran los 31 caracteres, se introdujo la instrucción END-DO para hacer más estructurados los loops, se introdujeron los punteros como nuevo tipo de dato y los módulos pasaron a formar parte de la nueva versión del lenguaje [4]. Hasta este momento todas las actualizaciones del estándar facultaban a los programadores a recompilar el código fuente escrito en versiones anteriores del mismo y éste era totalmente operativo en la nueva versión del lenguaje.

Hacia el año 1995 una revisión menor al estándar de Fortran 90 fue publicada; esta versión es conocida como Fortran 95 [18]. Ésta fue la única versión del lenguaje de programación que suprimió alguna de las caraterísticas obsoletas que figuran en el apéndice B del estándar, rompiendo así con una larga tradición mantenida entre las distintas versiones.

Aunque se pudiera pensar en este punto que Fortran ya no tenía más utilidad o aplicación como lenguaje de programación, hacia fines de 2004 [19] el comité de estandarización de Fortran decidió revisar nuevamente el estándar, publicando una nueva versión. Esta versión, conocida como Fortran 2003, incorporaba nuevas características como ser: mecanismos de programación orientada a objetos, tipos de datos parametrizados, interoperabilidad con el lenguaje C, punteros a funciones, etc.

Para fines de 2010, se espera que sea publicada una nueva revisión del estándar de Fortran, denominada Fortran 2008, en el cual se incorpora un modelo de procesamiento paralelo (co-arrays).

Fortran se ha convertido a lo largo de sus 50 años de vida en uno de los lenguajes de programación más utilizados dentro del ámbito científico. Éste aun se utiliza en áreas de las Ciencias de la Computación como por ejemplo High Performance Computing. Además lo utilizan muchas de las disciplinas denominadas ciencias duras como: Matemática, Física, Química, Estadística, Biología, Metereología, Geología., etc. Inclusive es utilizado también como herramienta en las Ciencias Económicas y Sociales.

4. Refactorización en Fortran

Nuestra propuesta es utilizar el proceso de refactorización para atacar el problema de código Fortran heredado. Existen en la actualidad muchos sistemas críticos escritos en versiones antiguas del lenguaje, que no sólo resultan difíciles de modificar sino simplemente muy difíciles de leer y entender. Nuestros refactorings proponen cambios que al mismo tiempo actualizan, la versión a otra de Fortran, mejoran cualidades internas del código como la legibilidad, mantenibilidad y extensibilidad.

A continuación se describirán cuatro problemas clásicos que presenta el código fuente heredado en Fortran y la solución a cada problema en términos de una refactorización. Estas transformaciones son propias del lenguaje y surgieron a partir de su peculiar evolución. Se utilizará como ejemplo el código de la Figura 1.

4.1. Cambiar de Formato Fijo a Formato Libre

Motivación: El formato fijo acompaña a Fortran desde sus comienzos [5,10]. Éste consiste en una línea de código fuente escrito en Fortran I, II, III, IV, 66 ó 77 que debe satisfacer el siguiente formato: en las columnas 1 a 5 se declaran las etiquetas, en la columna 6 se indicarán las continuaciones de línea, de la columna 7 a la 72 se debe escribir código Fortran y de la línea 73 en adelante se escribirán comentarios opcionalmente. Esta restricción en la escritura de un programa Fortran que dura más de 36 años, hace que se encuentre código fuente difícil de leer y comprender, sin un seguimiento exhaustivo del mismo (ver Figura 1). Un punto importante a tener en cuenta es que el formato fijo permite espacios ya sea en medio de una instrucción o de un nombre de una variable, por ejemplo "D O100I=1 ,10" en lugar de "DO 100 I = 1, 10". Esta propiedad del lenguaje tuvo relevancia, por ejemplo, en el fracaso de la zonda espacial MARINER I [30].

Solución: Aunque esta mejora podría ser realizada manualmente, resulta inviable hacerlo en el caso de grandes aplicaciones. Un segundo intento podría ser utilizando una herramienta que manipule el texto caracter a caracter y genere código "pretty-printed". El problema con este enfoque está centrado en que un

Figura 1. Ejemplo de código Fortran antes y después de las transformaciones propuesta
.

```
----|----1----|----2----|----3
      integer::temp
      integer::sorted
      integer::i
      integer::pass
      integer::j
      integer::k
      integer::w
  800 pass = 0
      ....
      ....
  999 if (pass) 1010,1001,1010
      pass=1
 1001 sorted = 1
      d o 1005 i = 1,count-pass
      if(data(i) .gt. data(i+1)) then
      t emp = data(i)
 1005 data(i) = data(i+1)
 1006 data(i+1) = temp
 1007 sorted = 0
      e ndif
 1008 continue
 1009 p ass = pass +1
      if(sorted .eq. 0) goto 1001
```

```
      integer::temp
      integer::sorted
      integer::i
      integer::pass


      pass = 0
     ....
     ....
      if (pass) 1010,1001,1010
      pass=1
1001  sorted = 1
      do 1005 i = 1,count-pass
          if(data(i) .gt. data(i+1)) then
              temp = data(i)
              data(i) = data(i+1)
              data(i+1) = temp
              sorted = 0
         endif
1005 continue
     pass = pass +1
     if(sorted .eq. 0) goto 1001
```

simple formateador de código no podría reconocer instrucciones y distinguir entre instrucciones o variables.

La solución que proponemos es utilizar la refactorización llamada *"Cambiar de Formato Fijo a Formato Libre"*, que implementa automáticamente esta transformación. Como explicaremos en la Sección 5 una herramienta de refactorización puede parsear correctamente las instrucciones del lenguaje y realizar las validaciones previas y posteriores para asegurar que el comportamiento sea preservado. Ésta ha sido unas de las refactorizaciones más requeridas por los usuarios de la herramienta Photran [2].

4.2. Eliminación de Variables Locales no Utilizadas

Motivación: Una práctica pobre de programación es declarar variables que no son utilizadas en ninguna parte del programa. Actualmente muchos IDE para otros lenguajesseñalan al programador cuando una variable no está siendo utilizada. Eliminar estas variables puede tornarse tedioso en grandes aplicaciones.

Solución: Eliminar todas aquellas variables que no son utilizadas dentro del ámbito donde fueron declaradas. Para ello es necesario conocer la cantidad de veces que se hace referencia a la misma en el código fuente. Si bien es posible realizar esta tarea manualmente, es sumamente tedioso y riesgoso. En la Figura 1 se puede apeciar un ejemplo de un antes y un después de la eliminación de variables no utilizadas.

El refactoring *"Eliminar Variables Locales no Utilizadas"* permite solucionar este problema automáticamente. La implementación de esta refactorización

requiere determinar cuáles son las variables declaradas, contar las referencias a cada una de ellas, y finalmente eliminar aquellas que no tienen referencias en el programa.

4.3. Eliminación de Etiquetas no Referenciadas

Motivación: En su proceso evolutivo Fortran tardó en adaptarse a la utilización del paradigma de programación estructurada. Esto demuestra que recién en la revisión de 1992 se introdujeron las instrucciones faltantes para poder confeccionar programas que se alinearan realmente a los conceptos de la programación estructurada [4]. Debido a este hecho, es frecuente la utilización de etiquetas como punto de llegada de saltos realizados desde muchas partes de un programa Fortran. En la Figura 1 se puede apreciar cómo estas etiquetas dificultan, no sólo la legibilidad, sino también comprensión del código escrito.

Solución: Para facilitar la limpieza del código y por ende mejorar la legibilidad y comprensión del mismo se deben retirar todas las etiquetas que ya no son más referenciadas por ninguna instrucción dentro del ámbito del programa. El mecanismo por el cual se cuentan las referencias de las etiquetas no es trivial. Para ello, al igual que en el caso anterior, es necesario tener un mecanismo que proporcione en forma certera cuáles son las etiquetas que no están referenciadas, descartando la modificación manual desde un principio.

Para ello se propone la refactorización *"Eliminar Etiquetas no Referenciadas"* en la cual se recolectarán por un lado todas las instrucciones etiquetadas y posteriormente todas las referencias a cada etiqueta. Una vez recorrido todo el programa se eliminarán las etiquetas que no sean referenciadas en ningún componente del programa.

4.4. Eliminación de las Instrucciones If Aritméticas

Motivación: Otra instrucción que forma parte del lenguaje hace ya más de 50 años es el IF aritmético. Esta instrucción es un formato primitivo de la conocida estructura de control If. Básicamente funciona con una variable que hace las veces de índice, el if aritmético compara el valor de la variable, que siempre debe de ser numérica. Dependiendo de si este valor es mayor que cero, menor que cero o igual a cero, la instrucción cede el flujo del programa a la etiqueta correspondiente.

Solución: Queda claro que el reemplazo de este tipo de construcción hace más legible y comprensible el código fuente. Siguiendo con el enfoque del artículo, la forma utilizada para resolver esta transformación es construir una equivalente a partir de una instrucción IF. Si bien un buen motor de "Search and Replace" podría ser de ayuda para alcanzar este objetivo, no es recomendable. Es necesario hacer explícitas las instrucciones GO TO subyacentes en el IF arimético, siendo un buen paso en la eliminación de los mismos. En la Figura 2 se puede apreciar la equivalencia entre ambas instrucciones.

Una solución acorde al concepto de refactorización es recorrer el programa y recolectar las construcciones que representen IF aritméticos y reemplazar los

Figura 2. Equivalencia entre un If Aritmético y un if Clásico.

```
if (x) 10,20,30
```

```
if(x< 0) then
    goto 10
else if(x == 0) then
    goto 20
else
    goto 30
end if
```

if aritméticos por su equivalente. Esta refactorización se denomina *"Quitar If Aritmético"*.

5. Una Herramienta de Refactorización para Fortran

En [21] se propone un catálogo de refactorizaciones para Fortran, algunas de las cuales están siendo implementadas en Photran. Photran es la única herramienta automatizada para refactorizar código fuente escrito en Fortran, que se encuentra actualmente en desarrollo [1].

Photran provee de una infraestructura de refactorización para poder implementar estas transformaciones. Escrito totalmente en Java, es una extensión de Eclipse CDT [11]. Además de ser una avanzada herramienta para refactorizar, Photran, es también un IDE para Fortran, que posee las características de los entornos de programación más avanzados, haciendo posible desarrollar y compilar programas en Fortran 66-2008. El desarrollo de Photran se originó en la Universidad de Illinois, pero actualmente es código libre del que participan varios grupos de trabajo.

La construcción de una herramienta de refactorización es muy compleja, porque requiere de una infraestructura para el análisis y la transformación que conserven el comportamiento del programa. El análisis de las precondiciones de cada refactorización (que aseguran la preservación del comportamiento) necesita ser además lo suficientemente rápido para que pueda realizarse interactivamente. Dados estos requerimientos, las herramientas de refactorización utilizan el árbol de sintaxis (AST) totalmente reescribible. En [25] se proporciona un método para generar estas estructuras de forma totalmente automática, utilizando el concepto de AST reescribible, que permite que sus nodos puedan ser reemplazados, reordenados, eliminados o agregados. Una vez transformado el AST se genera el código fuente correspondiente utilizando la técnica de "pretty-printing". Esta infraestructura es provista por Photran.

Cada refactorización implementada en Photran es una clase Java a ser extendida. Existen dos tipos de refactorizaciones: los "Editor Refactorings" permiten refactorizar un único archivo fuente o una selección del mismo, y los "Resource Refactorings", que permiten refactorizar un grupo completo de archivos o una selección de archivos. Los dos tipos de refactorizaciones se caracterizan por tener cuatro métodos que deben ser sobrescritos. Para crear un nuevo refactoring hay que, en primer lugar, proporcionar el nombre de la refactorización; en segundo

lugar hay que verificar las precondiciones iniciales para aplicar la refactorización y si fuera necesario solicitar información al usuario. Posteriormente, hay que chequear las condiciones finales requeridas por la transformación y finalmente realizar la transformación, que consistuye la modificación del AST que representa al programa, con la posterior persistencia del código fuente ya refactorizado.

En el caso del refactoring "Cambiar del Formato Fijo al Formato Libre" Sección 4.1, la herramienta nos asegura que todas las instrucciones serán reconocidas correctamente, pudiéndose distinguir entre una variable y una instrucción con o sin espacios de por medio. Las refactorizaciones propuestas en las Secciones 4.2 y 4.3 utilizan el patrón de diseño "Visitor" [13] para contar las referencias a cada variable (o etiqueta), eliminando del AST aquellas que no sean referenciadas.

6. Trabajos Relacionados

La restructuración de código fuente mediante la aplicación de transformaciones batchs existe ya desde hace años. Un ejemplo de ello es DMS [6], una herramienta que posibilita la aplicación de conceptos de reingeniería en programas escritos en distintos lenguajes de programación.

En lo que respecta a la aplicación específica para código fuente escrito en Fortran, Greenough y Worth [17] proponen varias herramientas para la aplicación de tranformación al código. Existen varias razones por las cuales estas herramientas no han sido exitosas. Una de ellas es la forma de aplicación, que por lo general es en modo batch sin interacción humana. Esto implica que las transformaciones aplicadas al código fuente no necesariamente facilitan la comprensión del mismo por parte del desarrollador.

Hacia 1992 se aplica el concepto de refactorización [23] en la producción de software reusable. Posteriormente y gracias a los aportes realizados por Ralph Johnson y su equipo en la Universidad de Illinois, y a los promotores de XP, el concepto de refactorización comenzó a extenderse. En [15,14] se propone por primera vez la utilización de la refactorización en el ámbito de un lenguaje de programación estructurada.

Posteriormente, se publica una serie de artículos en los cuales se analiza la posibilidad de aplicar refactorización en todas las versiones de Fortran [26,27]. En estos trabajos se plantea la necesidad de una herramienta de refactorización automatizada, a partir de la cual comienza el desarrollo de Photran. En [29,28] se propone una serie de transformaciones con el objetivo de mejorar software heredado en Fortran. Varias de estas transformaciones coinciden con las propuestas por Overbey et al.

En Méndez et al. presentamos el primer catálogo de refactorizaciones para Fortran, que incluyen desde su versión de Fortran 66 hasta la Fortran 2008 [21]. Allí proponemos además una clasificación de refactorizaciones, en `www.fortranrefactoring.com.ar` se puede encontrar publicada esta clasificación en detalle.

7. Conclusiones y Trabajos Futuros

Fortran en su larga vida como lenguaje de programación aún sigue vigente pese a ser presa de prejuicios. El espectro de aplicaciones que aún sigue ejecutándose y ha sido escrito en este lenguaje es significativamente importante.

Si bien se han utilizado herramientas automatizadas con el fin de mejorar los procesos de mantenimiento, implantación o migración de sistemas heredados escritos en Fortran, éstas no han tenido resultado en la contienda con este tipo de software. En este artículo se propone aplicar el concepto de refactorización como una herramienta que puede tener un profundo impacto en estos procesos relacionados con software heredados, además las transformaciones aplicadas al código fuente con el objetivo de mejorar su estructura interna, deben preservar el comportamiento externo del software y por lo tanto, la aplicación de conceptos como los AST reescribibles ayudan en este proceso.

Trabajos futuros buscarán recabar datos estadísticos con los cuales se pueda medir el grado de utilidad, la aplicabilidad y las dificultades en aplicar estas refactorizaciones. Se buscará determinar si alguna de estas refactorizaciones es una composición de otras menos complejas. Además sería de enorme importancia realizar un análisis estadístico de las características que presenta el código fuente heredado escrito en Fortran. Esto ayudaría a determinar cuál es la necesidad real de los programadores. Es de vital importancia generar una herramienta automatizada que provea bases que puedan ser extendidas para ser aplicadas en otros lenguajes de programación.

Referencias

1. Photran, an Integrated Development Environment and Refactoring Tool for Fortran. http://www.eclipse.org/photran/.
2. photran Users Mailing List. https://dev.eclipse.org/mailman/listinfo/photran.
3. American National Standards Institute. X3. 9-1978. *American National Standards Institute, New York*, 1978.
4. American National Standards Institute. *American National Standard for programming language, FORTRAN — extended: ANSI X3.198-1992: ISO/IEC 1539: 1991 (E)*. American National Standards Institute, September 1992.
5. J. Backus. The History of Fortran I, II, and III. *ACM SIGPLAN Notices*, 13(8):165–180, 1978.
6. I. Baxter, P. Pidgeon, and M. Mehlich. DMS: Program Transformations for Practical Scalable Software Evolution. In *Proceedings of the International Conference on Software Engineering, IEEE Press*, 2004.
7. K. Bennett. Legacy systems: Coping with success. *IEEE Software*, 12(1):19–23, 1995.
8. M.L. Brodie and M. Stonebraker. *Migrating legacy systems*. Morgan Kaufmann Publishers, 1995.
9. F.P. Brooks. No silver bullet: Essence and accidents of software engineering. *IEEE computer*, 20(4):10–19, 1987.
10. A. FORTRAN. X3. 9-1966. *American National Standards Institute Incorporated, New York*, 1966.

11. The Eclipse Foundation. Eclipse.org home. http://www.eclipse.org/, 2010.
12. M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 1999.
13. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison-Wesley Reading, MA, 1995.
14. A. Garrido. *Program Refactoring in the Presence of Preprocessor Directives*. PhD thesis, 2005.
15. A. Garrido and R. Johnson. Refactoring C with Conditional Compilation. In *18th IEEE International Conference on Automated Software Engineering (ASE 2003)*. IEEE Computer Society, 2003.
16. N.E. Gold. *The meaning of legacy systems*. Univ. of Durham, Dept. of Computer Science, 1998.
17. C. Greenough and D. Worth. The Transformation of Legacy Software: Some Tools and a Process. Technical report, RAL Technical Report TR-2003 012, 2004.
18. ISO. ANSI/ISO/IEC 1539-1:1997: Information technology — programming languages — Fortran — part 1: Base language.
19. ISO. ANSI/ISO/IEC 1539-1:2004(E): Information technology — Programming languages — Fortran Part 1: Base Language. pages xiv + 569, May 2004.
20. Méndez, Mariano. Refactoring de código estructurado, Trabajo de Especialización , Marzo 2010.
21. Méndez, Mariano and Overbey, J. L. and Garrido, A. and Tinetti, F. G. and Johnson, R. . A Catalog and Classification of Fortran Refactorings. *11th Argentine Symposium on Software Engineering (ASSE 2010)*, 2010.
22. T. Mens and T. Tourwé. A survey of software refactoring. *IEEE Transactions on software engineering*, 30(2):126–139, 2004.
23. W.F. Opdyke. *Refactoring Object-Oriented Frameworks*. PhD thesis, 1992.
24. W.F. Opdyke and R.E. Johnson. Refactoring: An aid in designing application frameworks and evolving object-oriented systems. In *Proceedings of Symposium on Object-Oriented Programming Emphasizing Practical Applications*, 1990.
25. J. Overbey and R. Johnson. Generating Rewritable Abstract Syntax Trees. *Software Language Engineering*, pages 114–133, 2009.
26. J. L. Overbey, S. Xanthos, R. Johnson, and B. Foote. Refactorings for Fortran and High-Performance Computing. In *SE-HPCS '05: Proceedings of the second international workshop on Software engineering for high performance computing system applications*, pages 37–39, New York, NY, USA, 2005. ACM.
27. J.L. Overbey, S. Negara, and R.E. Johnson. Refactoring and the Evolution of Fortran. In *2nd International Workshop on Software Engineering for Computational Science and Engineering (SECSE'09)*, 2009.
28. F. G. Tinetti, M. A. López, and P. G. Cajaraville. Fortran Legacy Code Performance Optimization: Sequential and Parallel Processing with OpenMP. *World Congress on Computer Science and Information Engineering*, pages 471–475, 2009.
29. Fernando G. Tinetti, Pedro G. Cajaraville, Juan C. Labraga, and Mónica A. López. Parallel Computing Applied to Climate Numerical Models (in Spanish). *IX Workshop de Investigadores en Ciencias de la Computación, Universidad Nacional de La Patagonia San Juan Bosco (UNPSJB)*, pages pp. 591–595, May 3-4, 2007. http://hpclinalg.webs.com/hpclinalg_en.html.
30. H.S. Tropp. FORTRAN anecdotes. *Annals of the History of Computing*, 6(1):61–62, 1984.