Thesis Overview

Fortran Refactorings for Legacy Systems

Mariano Méndez
Facultad de Informática
Universidad Nacional de La Plata
Agosto 2011
marianomendez@gmail.com

In 1956, the first draft of The IBM Mathematical Formula Translating System was finished. This first version of Fortran (called FORTRAN I) was the start of a complex evolutionary process. This process led to many different versions of the Fortran language, each of them with features required by the historical moment. Various features were incorporated during its evolution: subprograms (FORTRAN 66), an improved set of control structures to support structured programming (FORTRAN 77), modules and pointers (Fortran 90/95), object-oriented capabilities (Fortran 2003), submodules and co-arrays (Fortran 2008). For this evolution to be practical, the backward compatibility with the older versions of the language was essential. Over so many years of evolution, program maintenance becomes challenging. Many operations can be written in three or four different ways.
The magnitude of maintenance tasks is increased not only by the evolution of versions, but also by the large amount of Fortran code in production and the importance of Fortran as a programming language in several disciplines such as meteorology, physics and mathematics.

Trapped by its own essence (changeability, conformity, intangibility and complexity) software has to deal with:

- Evolution: Like other human products, software has to change according to people’s needs.
- Redesign: Unlike other human products, software can be modified even when its development process has been finished. If we take a closer look at manufactured products, like a pen: once it has been produced, the pen can not be changed and it will remain the same until the end of its days.
- Quality: As a consequence of its evolution and redesign, the software quality will be undermined.

As a successful programming language Fortran is characterized by a long lifetime and by having a huge production of legacy code due to its particular evolutionary process. Such process in which backward version compatibility is maintained and features deletion rarely occurs makes Fortran a very illustrative case to be studied.

There is not a formal definition of what a Legacy System is. However, we can find different approximations about what Legacy Systems are. Nicolas Gold, summarizes the Legacy System concept as follows: “Legacy Software is critical software that cannot be modified efficiently”. There is an aspect where legacy software becomes a challenge, it is the maintenance stage. In this stage, the software that has been running in production for 20 or 30 years is hard to manage because software gradually deteriorates. During the maintenance a program may need different types of changes. Enhancements, corrections, adaptations and preventions to a system may be needed. All of these tasks require knowledge and comprehension about the system. It becomes another aspect where legacy becomes also a challenge.

Refactoring is a technique used to improve internal qualities of the code like readability, flexibility, understandability and maintainability. It is applied interactively on code with "bad smells" like duplication and lack of parameters, and after a series of small transformations, it beautifies the code preserving its behavior. Using refactoring, developers and maintainers can manage the code and then extend it with new functionality. In the case of Fortran, refactoring can make substantial improvements to readability and maintainability, and it can also modernize the code by replacing obsolete constructs with newer alternatives. Moreover, refactoring may be used to improve external qualities like performance, which is highly beneficial in the case of Fortran since it is used mostly for high performance computing. However, the goals of high performance many times seem to oppose other goals of software engineering. For example, a transformation like loop unrolling will definitely worsen readability and maintainability of the code. The focus is not on these kinds of compiler optimizations, but on refactorings that improve maintainability as well as maintain high performance, and vice versa. Even more, refactorings that improve readability, an internal quality, may be applied to gain a better understanding of the program and then improve performance by parallelization.

This thesis is based on a certain type of legacy software that came from scientific research. Scientists have become one of the most important legacy code producers for many reasons. One of these is long-lived field (about 50 years old) they have been working in. Another reason is the amount of code produced through years and the lack of a well-defined software development process.

Even though refactoring concept was born within the pale of object oriented programming we think that this concept is a paramount tool to be applied on Fortran source code. Since it has been successfully used in C language, our objective was to build a reference catalog which will serve as a guide to Fortran programmers. In thesis we discussed and proposed a detailed catalog of Fortran source transformation.

Fortran evolution has resulted in a wide range of equivalent syntactical constructions. From those equivalent constructions, the older ones (coming from old language version/s) have many disadvantages/drawbacks. Programmers do not need to be aware of all these variations and/or Fortran's dialects in an academic course about Fortran programming, but the scenario radically changes if a programmer is working on a twenty-year-old application that has been written by others in FORTRAN 77. However, not all Fortran code is legacy code. Fortran has gained a leading role in the High Performance Computing world throughout the years. High Performance Fortran is an extension of Fortran 90 that supports parallel/vector computing. Co-Array Fortran is an extension of Fortran 95 supported by Cray compilers. Currently, old Fortran programs need to be made more efficient in multiprocessing systems with multi-core architectures. Furthermore, multi-core processors are making single-threaded (or, directly, sequential) software obsolete, such as most of the legacy Fortran programs.

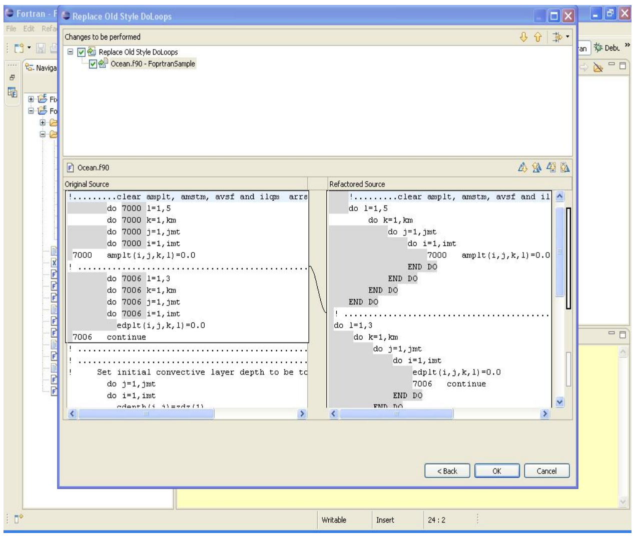
Photran is an advanced, multiplataform integrated development environment (IDE) for Fortran based on Eclipse. The tool has a number of powerful features. As an IDE, it integrates editing, source navigation, compilation, and debugging into a single tool. It uses makefiles for compilation, which allows it to work with virtually any existing Fortran compiler; so-called error parsers are provided which interpret the error messages from popular compilers, associating error markers with the appropriate lines of code. Language-based searching allows a Fortran programmer to quickly find a subprogram or module with a particular name, or to find all of the references to a particular variable or subprogram. From the beginning, Photran was designed to support refactoring, and much of its development effort has focused on providing a robust refactoring infrastructure. Version 6.0 (released June, 2010) contains 16 refactorings, and many more are under development. The actual version of Photran (released in June, 2011) contains 31 implemented refactorings from the proposed catalog. Four Fortran specific refactorings have been selected from the proposed catalog with the intention of describing and implementing them in this thesis.

Refactorings to Improve Maintainability
The refactorings in this category are intended to improve internal quality attributes of the code such as: readability, understandability and extensibility (attributes that refactoring has been recognized to improve) and also refactorings that allow upgrading the code to newer versions of Fortran, removing obsolete features:

- Refactorings to Improve Presentation / Readability.
- Refactorings to Facilitate Design/Interface Change.
- Refactorings to Avoid Poor Fortran Codding Practices.
- Refactorings to Remove Outdated,Obsolete and Non-Standard Constructs.

Performance Refactorings
This category currently has two examples of how refactoring can be used to improve performance while preserving not only the behavior of the program but also the readability and maintainability of the code. This is one of the factors that sets refactoring apart from optimization.

In the interest of understanding software improvement processes, it is natural for us to try to characterize the aspects of software that are affected in those processes. While the definition of a new readability or comprehensibility metric goes beyond the scope of this thesis, some kind of measurements are needed to be able to quantify the improvements achieved. One of the most important aspects that we aim to stress is that it is impossible to improve the design of unreadable source code. This unreadiness emerges from all those old and obsolete features of the language still valid, even in Fortran 2008. That is why refactoring is needed first to make the code readable. We define our magnitude as "Fortran Code Readability/Comprehensibility Scale" (FCRCS). This scale can be applied to a program, a module or a subroutine.

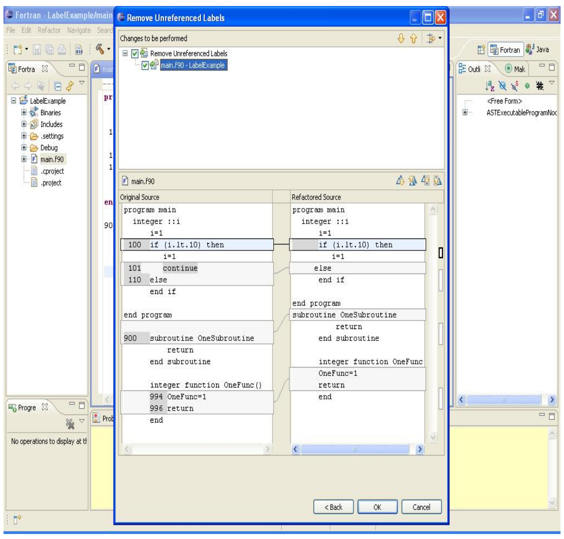


Original Source:

```
!.........clear amplt, amstm, avsf and ilqm  arra
         do 7000 l=1,5
         do 7000 k=1,km
         do 7000 j=1,jmt
         do 7000 i=1,imt
 7000    amplt(i,j,k,l)=0.0
! ...............................................
         do 7006 l=1,3
         do 7006 k=1,km
         do 7006 j=1,jmt
         do 7006 i=1,imt
          edplt(i,j,k,l)=0.0
 7006    continue
! ...............................................
! ...............................................
!     Set initial convective layer depth to be tc
         do j=1,jmt
         do i=1,imt
```

Refactored Source:

```
 !.........clear amplt, amstm, avsf and il
 do l=1,5
     do k=1,km
         do j=1,jmt
             do i=1,imt
                 7000   amplt(i,j,k,l)=0.0
             END DO
         END DO
     END DO
 END DO
! .............................................
do l=1,3
   do k=1,km
       do j=1,jmt
           do i=1,imt
               edplt(i,j,k,l)=0.0
               7006   continue
           END DO
```



Original Source:

```
program main
  integer ::i
      i=1
 100  if (i.lt.10) then
         i=1
 101     continue
 110  else
      end if

end program

900   subroutine OneSubroutine
          return
      end subroutine

      integer function OneFunc()
      994 OneFunc=1
      996 return
      end
```

Refactored Source:

```
program main
  integer ::i
      i=1
      if (i.lt.10) then
         i=1
    else
      end if

end program
subroutine OneSubroutine
          return
      end subroutine

      integer function OneFunc
      OneFunc=1
      return
      end
```

The major general contributions, independent of the previous example, are:

- A Classification of Fortran refactorings: The way in which the refactorings were proposed is the result of how we think programmers need to use refactoring in their daily work. So we present the refactorings classified from the programmer's point of view.
- A Detailed catalog of Fortran refactorings: Each refactoring proposed in this catalog has emerged from the Fortran programmer's needs. Our description rests on each refactoring motivation.
- A proposal of refactorings for parallelizing and performance improvements: For some of these refactorings it has been proved that a much better performance existed \cite{rodrigues}. A set of these transformations are closely related to those conducted by compilers to improve performance, like loop fusion or loop fission \cite{digrefactoring}.
- A specification of some refactorings: The implementation of a set of refactorings was explained in detailed and documented with the aim of providing a guide to be used in the initial steps in the refactoring built process.
- The use of refactorings on Fortran legacy systems: In this work we have shown how to employ refactorings in the field of legacy systems. Furthermore, we have used refactoring applied to one of the most long-lived programming language such as Fortran.
- A metric definition: We have presented a way to measure the source code transformation impact on source code readability as a metric called ``FCRCS''.
- A Contribution to Photran Project: The refactorings implemented in this thesis will be all included in Photran 7.0 release.
- A public web site containing the catalog in different languages: Aligned with the aims of this research, a public access web site was created to integrate and to promote Fortran refactorings and the eclipse-based-refactoring tool (Photran). This site was published in July 2010 \cite{FortranRefactoringWeb}.

Mariano Méndez
marianomendez@gmail.com