2017 IEEE/ACM 12th International Workshop on Software Engineering for Science (SE4Science)

WIP: Generating Sequence Diagrams for Modern Fortran

Anawat Leatongkam, Aziz Nanthaamornphong
Faculty of Technology and Environment
Prince of Songkla University, Phuket Campus
Phuket, Thailand
Email: anawat.le@phuket.psu.ac.th, aziz.n@phuket.psu.ac.th

Damian W. Rouson
Sourcery Institute
Berkeley, California, USA
Email: damian@sourceryinstitute.org

***Abstract*—Fortran finds widespread use in scientific and engineering communities that embraced computing early, including weather and climate science and mechanical, nuclear, and aerospace engineering. Over its lifetime, Fortran has evolved to support multiple programming paradigms, including Object-Oriented Programming (OOP). Despite the recently burgeoning ecosystem of tools and libraries supporting modern Fortran, there remains limited support for generating common Object-Oriented Design (OOD) diagrams from Fortran source code. ForUML partially fills this need by reverse engineering Unified Modeling Language (UML) class diagrams from object-oriented (OO) Fortran programs. Class diagrams provide useful information about class structures and inter-relationships, but class diagrams do not convey the temporal information required to understand runtime class behavior and interactions. UML sequence diagrams provide such important algorithmic details. This paper proposes to extend ForUML to extract UML sequence diagrams from Fortran code and to offer this capability via a widely used open-source platform. The paper argues that the proposed capability can raise the level of abstraction at which the computational science community discusses modern Fortran.**

***Keywords*-Object-oriented design; modern Fortran; reverse engineering; computational science; object-oriented programming**

I. Introduction

Numerous tools exist to visualize software structures and behavior. Such tools range in sophistication from syntax-aware editors to documentation generators and performance analyzers. Reverse engineering tools, for example, extract information directly from source code to generate diagrams that aid in understanding the system architecture [1]. The variety of available tools varies with implementation languages.

Building on previous reverse engineering work, the second author developed the ForUML [2] reverse engineering tool to extract class information from OO Fortran code to generate UML diagrams depicting class attributes, methods, and class relationships such as aggregation and inheritance. UML is a software- and hardware-design specification language.

ForUML's initial focus on class diagrams helped to put Fortran on more even footing with other popular OOP languages such as Java, C++. The ability to generate class diagrams also enhances programming pedagogy. In university settings, the authors have found that teaching Fortran as an OO language helps to bridge the gap between Fortran and students' experiences in other modern languages, the majority of which are OO.

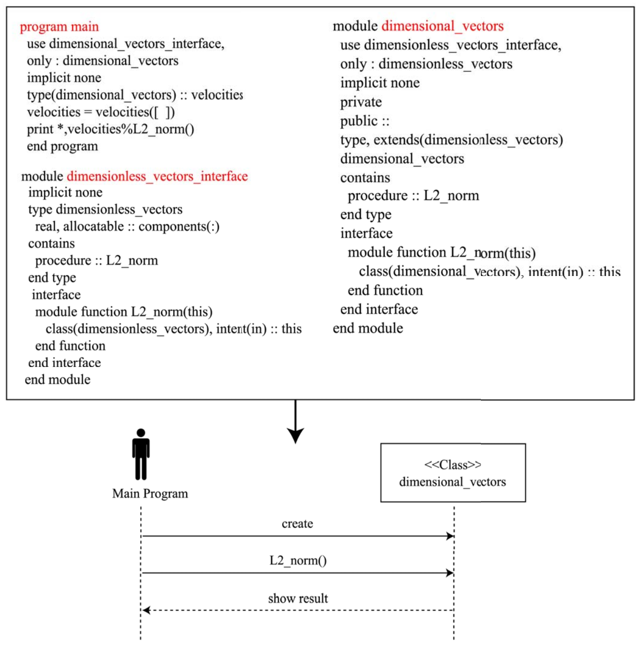
Class diagrams depict only static information: they do not explain the sequences of events that happen at runtime. Some other UML diagrams capture system dynamics [3], [4]. Enhancing design diagrams with dynamic concepts imparts a richer understanding of software systems. For example, software engineers use sequence diagrams to understand how objects interact with each other in a specific use case. A sequence diagram shows such interactions in the order that those interactions occur. Sequence diagrams may also convey the transition from requirements described as use cases to the next, more formal level of refinement.

We therefore propose to enhance ForUML to extract UML sequence diagrams from OO Fortran programs. We argue that having multiple design views allows developers to better analyze and understand complex software. We further argue that software maintainers can use different diagrams to capture different aspects of the original design concept to inform decisions around application's evolution under ongoing maintenance. Capturing the original designers' intentions can be especially important due to the long lifetimes of many scientific research applications [5]. Producing design diagrams automatically is especially important given that most developers of scientific applications are not software engineers and do not see the software itself as their primary work product.

II. The Proposed Solution

We propose that the automatic extraction of UML sequence diagrams will enhance developer interactions and language instruction by providing information that has long been recognized as useful but cannot currently be generated with any existing tool. Some of the relevant information has traditionally been captured in flowcharts for procedural code. However, flowcharts lack abstractions tailored to depicting object-oriented class behaviors and interactions. By UML sequence diagrams contain such information in addition to the temporal structure conveyed in flowcharts. Our implementation plan follows:

1. Design the rules of transforming Fortran code elements to sequence diagram notations. The third author's textbook provides examples of mapping object-oriented



```
program main
  use dimensional_vectors_interface,
  only : dimensional_vectors
  implicit none
  type(dimensional_vectors) :: velocities
  velocities = velocities([  ])
  print *,velocities%L2_norm()
  end program

module dimensionless_vectors_interface
  implicit none
  type dimensionless_vectors
    real, allocatable :: components(:)
  contains
    procedure :: L2_norm
  end type
   interface
    module function L2_norm(this)
       class(dimensionless_vectors), intent(in) :: this
    end function
  end interface
 end module
```

```
module dimensional_vectors
  use dimensionless_vectors_interface,
  only : dimensionless_vectors
  implicit none
  private
  public ::
  type, extends(dimensionless_vectors)
  dimensional_vectors
  contains
    procedure :: L2_norm
  end type
  interface
    module function L2_norm(this)
      class(dimensional_vectors), intent(in) :: this
    end function
  end interface
end module
```

Fig. 1. Example of Sequence Diagram Extracted from Fortran

Fortran code to UML sequence diagrams, including object creation, method invocation, and expression evaluation [6]. The mapping process employed in the book, however, was manual and not captured in any formal way. We are therefore developing new rules based on the Object Management Group (OMG) standard [7], which defined the specification of all UML diagrams. Initially, we designed the rules to cover the following UML sequence specification.

- Lifeline creations
- Lifeline's messages
- Messenger objects
- Receiver objects
- Conditional notations
- Decision notations
- Loop notations
- Coarray notations

These rules are under development and not yet completed. Fig. 1 illustrates the sequence diagram extracted from OO Fortran code, which each element and its relationship are considered under the rules.

2. Fortran code extractor. Once the rules are developed, we will develop the extractor module. This module can expose the elements and their relationships existing in given Fortran code files. These elements will be created to a tree node structure, which each tree node represents a class in Fortran. The extractor then applies the developed rules to the tree node and traverse all nodes in all trees. Before starting this process, the extractor will validate the Fortran grammar. Only the validated code will proceed.

3. XMI Generator. The transformation process will generate the output as an XML Metadata Interchange (XMI) file, which is the same output format as the original ForUML version. This XMI file contains the necessary information for representing a UML sequence diagram. Finally, the XMI file will be imported into an existing UML modeling tool for visualizing the diagram. Many tools support XMI [8], but the XMI functionality of many tools do not comply with the XMI standard. Thus, we will evaluate several UML modeling tools. ForUML currently launches ArgoUML [9] for diagram visualization.

In addition to developing the UML sequence diagram generation function, we plan to add following features into ForUML.

- Generate Fotran code form UML class diagrams
- Support drag & drop files into the ForUML window
- More complete descriptions of parallel data structures ("coarrays"), showing the number of (local) dimensions and the number of (global) codimensions

We will test the new version with existing OO Fortran packages. We will also evaluate user satisfaction via questionnaires with the goal to improve the tool in term of usability and correctness.

To encourage wider adoption and use of ForUML, we have released the current version of ForUML as open-source software on GitHub. Opening the source expands the opportunities for feedback and community contributions. Demonstrating the practical utility of ForUML for large-scale scientific programming will make it a valuable contribution to the software engineering and scientific communities.

III. Conclusion

This work primarily targets scientific software developers writing OO Fortran. We believe the updated capability of ForUML will provide benefits to the scientific developer communities as follows: (1) having high-level visual abstractions can increase productivity by reducing the learning curve and training time for applying software engineering practices in scientific software development projects and (2) having multiple views in UML diagrams will support software maintenance throughout development. However, these claims must be empirically evaluated in the future.

Acknowledgment

The authors would like to thank the ForUML users who have provided the valuable feedback.

References

[1] E. J. Chikofsky and J. H. Cross II, "Reverse engineering and design recovery: A taxonomy," *IEEE Softw.*, vol. 7, no. 1, pp. 13–17, Jan. 1990.
[2] A. Nanthaamornphong, J. Carver, K. Morris, and S. Filippone, "Extracting uml class diagrams from object-oriented fortran: Foruml," *Scientific Programming*, vol. 2015, 2015.
[3] M. Shirole and R. Kumar, "Testing for concurrency in uml diagrams," *SIGSOFT Softw. Eng. Notes*, vol. 37, no. 5, pp. 1–8, Sep. 2012.
[4] G. Booch, J. Rumbaugh, and I. Jacobson, *Unified Modeling Language User Guide, The (2Nd Edition) (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2005.
[5] J. Segal, "Some challenges facing software engineers developing software for scientists," in *Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, 2009, pp. 9–14.
[6] D. Rouson, J. Xia, and X. Xu, *Scientific software design: the object-oriented way*. Cambridge University Press, 2011.
[7] I. Object Management Group, "Object Management Group (OMG)," http://www.omg.org, 1997, accessed December 2013.
[8] Wikipedia, http://bit.ly/wikipedia-list-of-UML-tools, 2017, accessed January 2017.
[9] CollabNet, http://argouml.tigris.org, 2001, accessed January 2017.