Extracting UML Class Diagrams from Object-Oriented Fortran: ForUML

Aziz Nanthaamornphong
Department of Computer Science
University of Alabama
Tuscaloosa, Alabama, USA
ananthaamornphong@ua.edu

Karla Morris
Sandia National Laboratories
7011 East Avenue
Livermore, California, USA
knmorri@sandia.gov

Salvatore Filippone
Department of Industrial Engineering
University of Rome
"Tor Vergata", Italy
salvatore.filippone@uniroma2.it

ABSTRACT

Many scientists and engineers who implement high performance computing (HPC) software have adopted the object-oriented (OO) Fortran paradigm. One of the challenges faced by OO Fortran developers is the inability to obtain high level software design descriptions of existing applications. Knowledge of the overall software design is not only valuable in the absence of documentation, it can also serve to assist developers with accomplishing different tasks during the software development process, especially maintenance and refactoring. The software engineering community commonly uses reverse engineering techniques to deal with this challenge. A number of reverse engineering-based tools have been proposed, but few of them can be applied to object-oriented Fortran applications.

In this paper, we propose a software tool to extract unified modeling language (UML) class diagrams from Fortran code. The UML class diagram facilitates the developers' ability to examine the entities and their relationships in the software system. The extracted diagrams enhance software maintenance and refactoring. The experiments carried out with the aim to evaluate the proposed tool show its accuracy and a few limitations.

Categories and Subject Descriptors

D.1.3 [**Programming Techniques**]: Concurrent Programming - Distributed Programming and Parallel Programming; D.2.2 [**Software Engineering**]: Design Tools and Techniques

General Terms

Design, Languages, Tools

Keywords

Fortran programming, Reverse Engineering, Computational Science and Engineering, High Performance Computing

SEHPCCSE'13, November 17 - 21 2013, Denver, CO, USA
Copyright 2013 ACM 978-1-4503-2499-1/13/11$15.00.
http://dx.doi.org/10.1145/2532352.2532354.

1. INTRODUCTION

Recently, the development of software for computational science and engineering (CSE) research has drawn increased attention from the software engineering (SE) community. Computational research has been referred to as the third leg of scientific and engineering research, along with experimental and theoretical research. CSE researchers develop software to simulate natural phenomena that for various reasons cannot be studied experimentally. HPC is an important constituent of CSE software that often requires processing large amounts of data and floating-point operations to solve physical problems of increasing complexity [6]. Studying the development of CSE software is important because it supports a number of critical application domains, including: weather forecasting, astrophysics, constructions of new physical materials, cancer research, and many others. The impact of CSE on society is large due to the criticality of the problems addressed by CSE [5].

In this critical type of software, Fortran has been the preferred programming language among the HPC and CSE communities [7, 15]. Due to the growing complexity of the problems being addressed through CSE, the procedural programming style available in a language like Fortran 77 is no longer sufficient. Many developers have applied the object-oriented programming (OOP) paradigm to effectively implement the complex data structures used within CSE software. In the case of Fortran developers, this paradigm was first emulated following an object-based approach in Fortran 90/95 [8]. The Fortran 2003 language standard includes full support of OOP constructs, and as such influenced the advent of several HPC and CSE packages [3, 10, 18, 22, 23].

One of the greatest challenges faced by CSE developers is the ability to effectively maintain their software [17]. Since most CSE projects have a life span of several years, this concern implies high development and maintenance costs during a software system's lifetime. The difficulty of the maintenance process is affected by at least two factors.

First, CSE developers have a strong background in theoretical science, but they often do not have formal training in SE techniques. Additionally, some SE tools are difficult to use in a CSE development environment [6]. Consequently, CSE developers often have trouble identifying and using the most appropriate SE techniques for their work, in particular as it relates to reverse engineering tasks. For example, Storey noted that CSE developers who lack formal SE training need help with program comprehension when they are developing complex applications [25]. To address this problem, the SE community must develop tools that satisfy the needs of CSE developers. These tools must allow developers to perform important reverse engineering tasks with simplicity. More specifically, a visualization-based tool is appropriate for program comprehension in complex object-oriented applications [21].

Second, CSE software typically lacks adequate development oriented documentation [24]. In fact, documentation for CSE software often exists only in the form of subroutine library documentation. This documentation is usually quite clear and sufficient for library users, who treat the library as a black box, but not sufficient for developers who need to understand the library in enough detail to maintain it. The lack of design documentation in particular leads to multiple problems. Newcomers to a project must invest a lot of effort to understand the code. There is an increased risk of failure when software developers do not understand correctly how to interact with the software system. Additionally, the lack of documentation makes refactoring difficult and error-prone. The refactoring specific to high-performance and parallel computing is an important issue, in particular the optimization [20]. The refactoring is typically used to improve the code and software design. To use refactoring strategies, such as shortening the methods, breaking a large methods into smaller methods, can improve the software quality (e.g., performance, scalability) [16]. The developers need to be able to identify where the code should be refactored. The availability of appropriate design documentation can reduce the likelihood of poor choices during the refactoring process. Therefore, a tool that supports the refactoring activities would be very helpful.

To help CSE developers overcome those challenges, we developed a tool, *ForUML*, for extracting UML class diagrams from OO Fortran code. The transformation process is based on a reverse engineering approach and uses a Fortran parser, which does not depend on any specific Fortran compiler to generate output in XML Metadata Interchange (XMI) format. The UML class diagram is represented by the UML modeling tool *ArgoUML*[1]. We evaluated the accuracy of *ForUML* using five CSE software packages that use object-oriented features in Fortran 95, 2003, and 2008.

Although a number of reverse engineering tools have been developed (see Section 2), those tools that can be applied to OO Fortran (e.g., Doxygen[2]), do not provide the full set of documentation required by developers. Therefore, we propose the *ForUML* tool to automatically reverse engineers the necessary design documentation based on UML diagrams from OO Fortran source code. *ForUML* will provide the following benefits to the HPC community who use OO Fortran (the target of our work): 1) the extracted UML class diagrams should support software maintenance and refactoring throughout the software development process; and 2) tool support should reduce training time and speed up the learning curve for applying SE practices in HPC software development. For instance, *ForUML* will help developers perform refactoring activities by allowing them to evaluate the results of the refactoring visually via UML diagrams rather than manually via code inspection. In combination with a profiling and performance tool, the proposed tool would also assist the developer in refactoring source code with poor performance, since the knowledge of the software structure will provide the developer with additional information that can be used to change object dependencies and method implementation.

[1]http://argouml.tigris.org/

[2]http://www.doxygen.org

The rest of this paper is organized as follows. Section 2 provides an overview of related work. Section 3 presents ForUML and its process. Section 4 discusses the evaluation of ForUML along with lesson learned. Finally, Section 5 draws conclusions and presents future work.

2. RELATED WORK

ForUML builds upon and expands some existing work. Timothy et al. [14] provide the schema for static structure of source code, called *The Dagstuhl Middle Metamodel (DMM)*. This schema is used to represent models extracted from source code written in most common object-oriented programming languages for reverse engineering tasks. We applied the idea of DMM to the object-oriented Fortran.

The basic idea of using an XMI file to maintain the metadata for UML diagrams was drawn from four reverse engineering tools. Alfi et al. developed two tools that use XMI to maintain the metadata for the UML diagrams: a tool that generates UML sequence diagrams for web application code [2] and a tool to create UML-Entity Relationship diagrams for the Structured Query Language (SQL) [1]. Similarly, Korshunova et al. [13] developed *CPP2XMI* to extract various UML diagrams from C++ source code. *CPP2XMI* generates an XMI document that describes the UML diagram, which is then displayed graphically by DOT (part of the Graphviz framework) [12]. Duffy et al. [9] created *libthorin*, a tool to convert C++ source code into UML diagrams. Prior to converting an XMI document into a UML diagram, *libthorin* requires developers to use a third party compiler to compile code into the DWARF[3] (a debugging file format is used to support source level debugging). In terms of Fortran, DWARF only supports Fortran 90, which does not include all object-oriented features. This limitation may cause compatibility problems with different Fortran compilers. Conversely, ForUML is completely compiler independent and able to generate UML for all OO Fortran code.

Doxygen is a documentation tool that can use Fortran code to generate either a simple, textual representation with procedural interface information or a graphical representation. The only OOP class relationship Doxygen supports is inheritance. With respect to our goals, Doxygen has two primary weaknesses. First, it does not support all OOP features within Fortran (e.g., type-bound procedure, component). Second, the diagrams generated by Doxygen only include class names and class relationships, but do not contain other important information typically included in UML class diagrams (e.g., methods, properties). Our work expands upon Doxygen by adding support for OO Fortran and by generating UML diagrams that include all relevant information about the included classes (e.g., properties, methods, and signatures).

There are a number of available tools (both open source and commercial) that claim to transform OO code into UML diagrams (e.g., Altova UModel®, Enterprise Architect®, StarUML, and ArgoUML). However, in terms of our work, these tools do not support OO Fortran. Although they can-

[3]http://www.dwarfstd.org

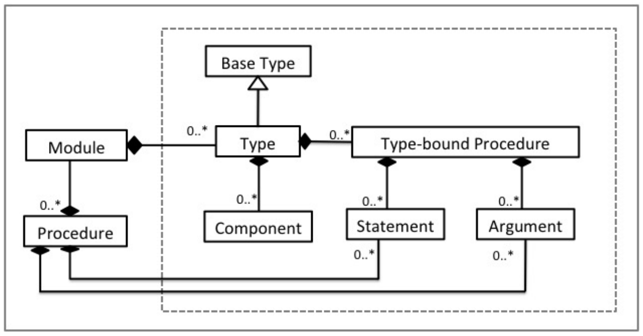


Figure 1: The Fortran model

not directly create UML diagrams from OO Fortran code, most of these tools are able to import the metadata describing UML diagrams (i.e. the XMI file) and generate the corresponding UML diagrams. *ForUML* can take advantage of this feature to display the UML diagrams described by the XMI files it generates separately from OO Fortran code.

This previous work has contributed significantly to the reverse engineering tools of traditional software. ForUML specifically offers a method to reverse engineer code implemented with modern Fortran, including features in the Fortran 2008 standard. Moreover, the tool was deliberately designed to support important features of Fortran, such as coarrays, procedure overloading, operator overloading.

3. ForUML

The primary goal of *ForUML* is to reverse engineer UML class diagrams from Fortran code. By extracting a set of source files, it builds a collection of objects associated with syntactic entities and relations. Object-based features were first introduced in the Fortran 90 language standard. Accordingly, *ForUML* supports all versions of Fortran 90 and later, which encompasses most platforms and compiler vendors. We implemented *ForUML* using Java Platform SE6 so that it could run on any client computing systems.

The UML object diagram in Figure 1 expresses the model of the Fortran language. The *Module* object corresponds to Fortran modules, i.e., containers holding *Type* and *Procedure* objects. The *Type-bound procedure* and *Component* objects are modeled with a composition association to instances of *Type*. Both the *Procedure* and *Type-bound procedure* objects are composed of *Argument* and *Statement* objects. The generalization relation with *Base Type* object leads to the parents in the inheritance hierarchy. When generating the class diagram in *ForUML*, we consider only the objects inside the dashed-line that separates object-oriented entities from the module-related entities.

Figure 2 provides an overview of the transformation process embodied in *ForUML*, comprising the following steps: *Parsing*, *Extraction*, *Generating*, and *Importing*. The following subsections discuss each step in more detail.

Parsing: The Fortran code is parsed by the Open Fortran Parser (OFP)[4]. We customized the OFP libraries to translate particular abstract syntax tree (AST) nodes (i.e., *Type*, *Component*, and *Type-bound procedure*) into objects. First, *ForUML* verifies the syntax in the source file and eliminates

[4]http://fortran-parser.sourceforge.net

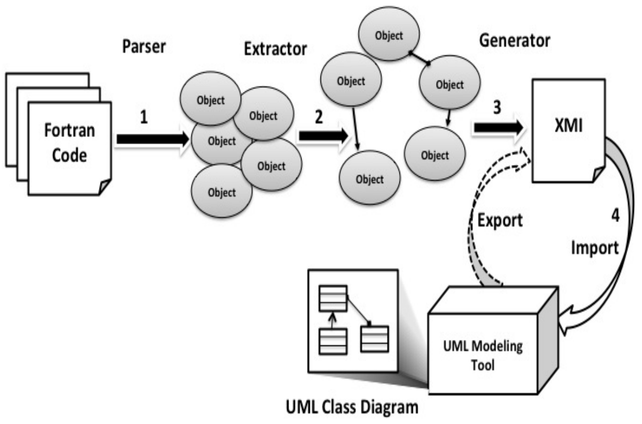


Figure 2: The Transformation Process

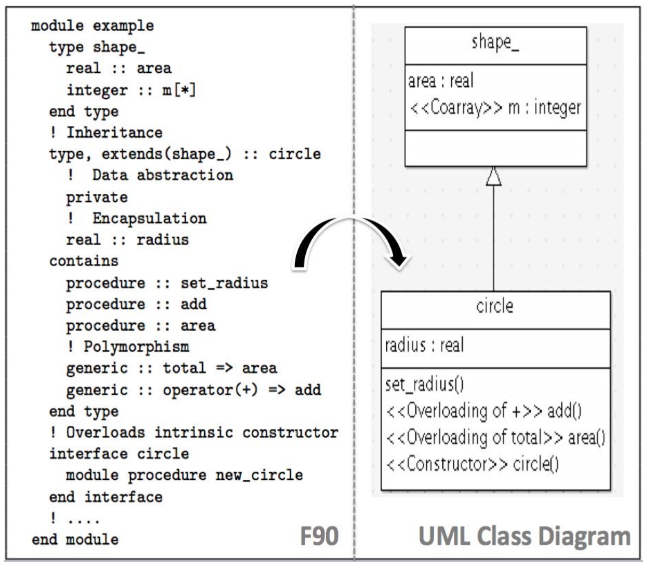
source files that have syntax problems. It also eliminates source files that do not contain any instances of *Type* or *Module*. *ForUML* reports the results of this step to the user via a GUI. Then, the parser extracts constructs occurring in the source code to build AST nodes. Note that *ForUML* only parses the selected input source files. Any associated *Type* objects that exist in files not selected by the user are not included in the class diagram.

Extraction: Next, *ForUML* extracts the objects. During the extraction, *ForUML* determines the type of relationship among the extracted objects. It maps each relationship to a specific relationship type object. *ForUML* supports two relationship types: *Composition* and *Generalization*.

- *Composition* represents the *whole-part* relationship. The lifetime of the part classifier depends upon the lifetime of the whole classifier. In other words, a composition describes a relationship in which one class is composed of many other classes. In our case, the *Composition* association will be produced when a *Type* object refers to another *Type* object in the *component*. However, the association that refers to the *Type*, which was not provided by the user does not appear in the class diagram. In the UML diagram, a composition relationship appears as a solid line with a filled diamond at the association end that is connected to the whole class.

- *Generalization* represents an *is-a* relationship between a general object and its derived specific objects, commonly known as an *Inheritance* relation. Similar to the *composition* association, the generalization association is not shown in the class diagram if the source file of the base type is not provided by the user. This relationship is represented by a solid line with a hollow unfilled arrowhead that points from the child class to the parent class.

Generating: We developed the XMI generator module to convert the extracted objects into XMI notation based upon our defined rules for mapping the extracted objects to the proper XMI notation. The rules for mapping the extracted objects and XMI document are specified in Table 1. In addition to these rules, we developed new stereotype notations for the constructor, coarray constructs, type-bound procedure overloading, operator overloading, such as `<<Constructor>>`, `<<Coarray>>`, `<<Overloading>>`, and `<<Overloading of +>>`. Figure 3 provides the sample Fortran code without procedure implementation (due to space constraints) and its generated class diagram including stereotypes.



```
module example
  type shape_
    real :: area
    integer :: m[*]
  end type
  ! Inheritance
  type, extends(shape_) :: circle
    !  Data abstraction
    private
    !  Encapsulation
    real :: radius
  contains
    procedure :: set_radius
    procedure :: add
    procedure :: area
    ! Polymorphism
    generic :: total => area
    generic :: operator(+) => add
  end type
  ! Overloads intrinsic constructor
  interface circle
    module procedure new_circle
  end interface
  ! ....
end module
```

Figure 3: Sample code snippet of Fortran supported by ForUML

Importing: To visually represent the extracted information as a UML class diagram, we import the XMI document into a UML modeling tool. We decided to include a UML modeling tool directly in *ForUML* to prevent the user from having to install or use a second application for visualization. We chose to include ArgoUML as the UML visualization tool in the current version of *ForUML*. We had to modify the ArgoUML code to allow it to automatically import the XMI document. Of course, if a user would prefer to use a different modeling tool, he or she can manually import the generated XMI file into any tool that supports the XMI format.

After importing the XMI file, ArgoUML's default view of the class diagram does not show any entities in the editing pane. Like the WYSIWYG[5] concept, the user needs to drag the target entity from a hierarchical view to the editing pane. To help with this problem, we added features so that ArgoUML will show all entities in the editing pane immediately after successfully importing the XMI document. Note that the XMI document does not specify how to present the elements graphically, so ArgoUML automatically adjusts the diagram when rendering the graphics. Each graphical tool may have its own method for generating the graphical layout of diagrams. The key reasons why we chose to integrate ArgoUML into *ForUML* are: 1) open source and implemented with Java making its integration seamless; 2) has sufficient documentation; and 3) provides sufficient basic functions required by the users (e.g., Export graphics, Import/Export XMI, Zooming).

ForUML provides a Java-based user interface for executing the command. To create a UML class diagram, the user performs these steps:

[5]WYSIWYG is acronym for "what you see is what you get"

Table 1: Fortran to XMI Conversion Rules

Fortran	XMI elements
Derived Type	UML:Class
Type-bound Procedure	UML:Operation
Dummy Argument	UML:Parameter
Component	UML:Attribute
Intrinsic type	UML:DataType
Parent Type	UML:Generalization.parent
Extended Type	UML:Generalization.child
Composite	UML:Association (the aggregation property as 'composite')

1. Select the Fortran source code;
2. Select the location to save the output; and
3. Open the UML diagram.

Figure 4 shows screenshots from the *ForUML* tool. Figure 4(a) illustrates how a user can select multiple Fortran source files for input to *ForUML*. The *Add* button opens a new window to select target file(s). Users can remove the selected file(s) by selecting the *Remove* button. The *Reset* button clears all selected files. After selecting the source files, the user chooses the location to save the generated XMI document (.xmi file). The *Generate* button activates the transformation process. During the process, the user can see whether each given source file is successfully parsed or not (Figure 4(b)). Once the XMI document is successfully generated, the user can view the class diagram by selecting the *View* button. Figure 4(c) illustrates the UML class diagram that is automatically represented in the editing pane with the ArgoUML. ArgoUML allows users to refine the diagram and then decide to either save the project or export the XMI document, which contains all the modified information.

4. EVALUATION AND DISCUSSION

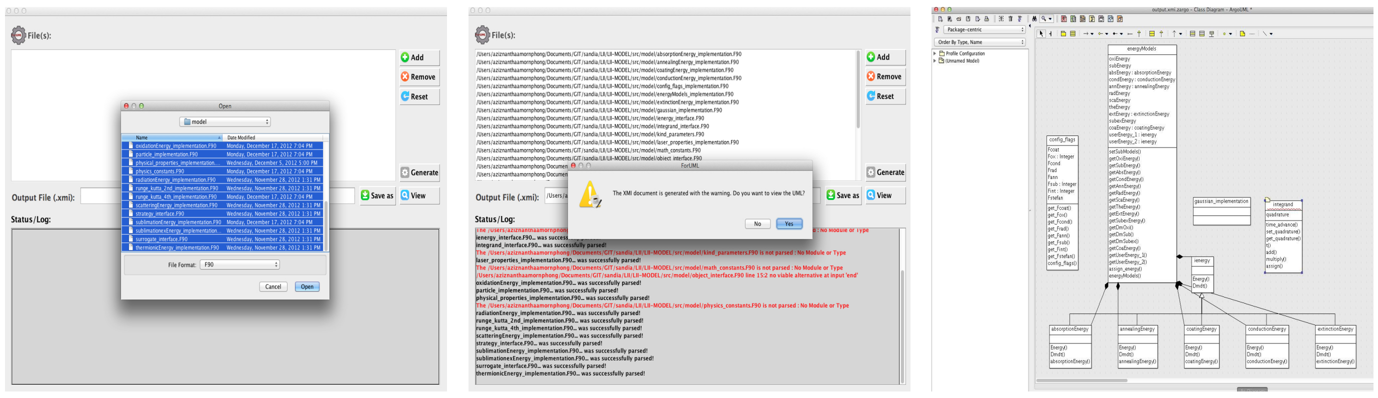
To assess the effectiveness of *ForUML*, we conducted some small experiments to gather data about its accuracy in extracting UML constructs from code. The following subsections provide the details of evaluations and results along with the limitations of *ForUML* and some lessons learned from the studies. The complete results of the evaluation on all packages and obtained class diagrams are provided in an accompanying website.[6]

4.1 Experimental Design

We evaluated the **accuracy** of *ForUML* on five OO Fortran software packages by adopting the definitions of *recall* and *precision* defined by Tonella et al. [26]:

- *Recall* measures the percentage of the various objects, i.e., Type, Components, Type-bound procedure, and Associations, in the source code correctly identified by *ForUML*.
- *Precision* measures the percentage of the objects identified by *ForUML* that are correct when compared with the source code.

[6]http://aziz.students.cs.ua.edu/foruml-eval.htm



Name	Date Modified
oxidationEnergy_implementation.F90	Monday, December 17, 2012 7:04 PM
particle_implementation.F90	Monday, December 17, 2012 7:04 PM
physical_properties_implementation....	Wednesday, December 5, 2012 5:00 PM
physics_constants.F90	Monday, December 17, 2012 7:04 PM
radiationEnergy_implementation.F90	Wednesday, November 28, 2012 1:31 PM
runge_kutta_2nd_implementation.F90	Wednesday, November 28, 2012 1:31 PM
runge_kutta_4th_implementation.F90	Monday, December 17, 2012 7:04 PM
scatteringEnergy_implementation.F90	Wednesday, November 28, 2012 1:31 PM
strategy_interface.F90	Wednesday, November 28, 2012 1:31 PM
sublimationEnergy_implementation.F90	Monday, December 17, 2012 7:04 PM
sublimationexEnergy_implementation...	Wednesday, November 28, 2012 1:31 PM
surrogate_interface.F90	Wednesday, November 28, 2012 1:31 PM
thermionicEnergy_implementation.F90	Wednesday, November 28, 2012 1:31 PM

```
/Users/aziznanthaamornphong/Documents/GIT/sandia/LII/LII-MODEL/src/model/absorptionEnergy_implementation.F90
/Users/aziznanthaamornphong/Documents/GIT/sandia/LII/LII-MODEL/src/model/annealingEnergy_implementation.F90
/Users/aziznanthaamornphong/Documents/GIT/sandia/LII/LII-MODEL/src/model/coatingEnergy_implementation.F90
/Users/aziznanthaamornphong/Documents/GIT/sandia/LII/LII-MODEL/src/model/conductionEnergy_implementation.F90
/Users/aziznanthaamornphong/Documents/GIT/sandia/LII/LII-MODEL/src/model/config_flags_implementation.F90
/Users/aziznanthaamornphong/Documents/GIT/sandia/LII/LII-MODEL/src/model/energyModels_implementation.F90
/Users/aziznanthaamornphong/Documents/GIT/sandia/LII/LII-MODEL/src/model/extinctionEnergy_implementation.F90
/Users/aziznanthaamornphong/Documents/GIT/sandia/LII/LII-MODEL/src/model/gaussian_implementation.F90
/Users/aziznanthaamornphong/Documents/GIT/sandia/LII/LII-MODEL/src/model/ienergy_interface.F90
/Users/aziznanthaamornphong/Documents/GIT/sandia/LII/LII-MODEL/src/model/integrand_interface.F90
/Users/aziznanthaamornphong/Documents/GIT/sandia/LII/LII-MODEL/src/model/kind_parameters.F90
/Users/aziznanthaamornphong/Documents/GIT/sandia/LII/LII-MODEL/src/model/laser_properties_implementation.F90
/Users/aziznanthaamornphong/Documents/GIT/sandia/LII/LII-MODEL/src/model/math_constants.F90
/Users/aziznanthaamornphong/Documents/GIT/sandia/LII/LII-MODEL/src/model/object_interface.F90
/Users/aziznanthaamornphong/Documen
/Users/aziznanthaamornphong/Documen
```

The XMI document is generated with the warning. Do you want to view the UML?

```
ienergy_interface.F90... was success
integrand_interface.F90... was successfully parsed!
The /Users/aziznanthaamornphong/Documents/GIT/sandia/LII/LII-MODEL/src/model/kind_parameters.F90 is not parsed : No Module or Type
laser_properties_implementation.F90... was successfully parsed!
The /Users/aziznanthaamornphong/Documents/GIT/sandia/LII/LII-MODEL/src/model/math_constants.F90 is not parsed : No Module or Type
/Users/aziznanthaamornphong/Documents/GIT/sandia/LII/LII-MODEL/src/model/object_interface.F90 line 15:2 no viable alternative at input 'end'
oxidationEnergy_implementation.F90... was successfully parsed!
particle_implementation.F90... was successfully parsed!
physical_properties_implementation.F90... was successfully parsed!
The /Users/aziznanthaamornphong/Documents/GIT/sandia/LII/LII-MODEL/src/model/physics_constants.F90 is not parsed : No Module or Type
radiationEnergy_implementation.F90... was successfully parsed!
runge_kutta_2nd_implementation.F90... was successfully parsed!
runge_kutta_4th_implementation.F90... was successfully parsed!
scatteringEnergy_implementation.F90... was successfully parsed!
strategy_interface.F90... was successfully parsed!
sublimationEnergy_implementation.F90... was successfully parsed!
sublimationexEnergy_implementation.F90... was successfully parsed!
surrogate_interface.F90... was successfully parsed!
thermionicEnergy_implementation.F90... was successfully parsed!
```

(a) Selection the Fortran Code (b) Generating the XMI (c) View the UML class diagram

Figure 4: Screenshots of ForUML

Table 2: Evaluation of ForUML : recall (extracted data / actual data)

Packages	Sub-Packages	Type	Procedure	Component	Relation	
					Inheritance	Composition
ForTrilinos	Epetra	16/16	304/304	17/17	12/12	2/2
	Aztecoo	1/1	12/12	1/1	0/0	0/0
	Amesos	1/1	7/7	1/1	0/0	0/0
	ForTrilinos	48/48	11/11	139/139	4/4	4/4
CLiiME	model	23/23	167/167	61/61	32/32	32/32
PSBLAS	modules	50/50	1309/1309	160/160	34/34	28/28
	prec	20/20	208/208	28/28	24/24	12/12
MLDP4	miprec	11/11	0/0	67/66	0/0	10/10
MPFlows	spray	10/10	55/55	29/29	2/2	3/3
	Overall	180/180 (100%)	2073/2073 (100%)	503/503 (100%)	108/108 (100%)	91/91 (100%)

We performed the evaluations as follows.

1. First, we manually inspected the source code to document the number of relevant objects in each package.

2. Second, we ran *ForUML* on each software package and documented the number of relevant objects included in the generated class diagram. *Note: we performed step one multiple times to ensure that the numbers were not biased by human error.*

3. Third, to compute *recall*, we compared the number of objects manually identified in the source code (Step 1) with the number identified by *ForUML* (Step 2).

4. Finally, to compute *precision*, we determined whether there were any objects produced by *ForUML* (Step 2) that we did manually observe in the code (Step 1).

4.2 Subject Systems

The five software packages we used in the experiments were 1) ForTrilinos[7]; 2) CLiiME; 3) PSBLAS[8]; 4) MLD2P4[9]; and 5) MPFlows. We selected these software packages because they were intentionally developed for using in the HPC environment. Two of the software packages (CLiiME and MPFlows) are not yet publicly available. A description of each software package follows.

[7]http://trilinos.sandia.gov/packages/fortrilinos/
[8]http://www.ce.uniroma2.it/psblas/
[9]http://www.mld2p4.it

4.2.1 ForTrilinos

ForTrilinos consists of an OO Fortran interface to expand the use of Trilinos into communities that predominantly write Fortran. Trilinos is a collection of parallel numerical solver libraries for the solution of CSE applications in the HPC environment. To provide portability, ForTrilinos extensively exploits the Fortran 2003 standard's support for interoperability with C. ForTrilinos includes 4 sub-packages (*epetra, aztecoo, amesos, and fortrilinos*), 36 files, and 36 modules.

4.2.2 CLiiME

Community Laser Induced Incandescence Modeling Environment (CLiiME) is a dynamic simulation model that predicts the temporal response of laser-induced incandescence from carbonaceous particles. CLiiME is implemented with Fortran 2003. It contains 2 sub-packages (*model* and *utilities*), 30 files, and 29 modules.

4.2.3 PSBLAS

PSBLAS 3.0 is a library for parallel sparse matrix computations, mostly dealing with the iterative solution of sparse linear system via a distributed memory paradigm. The library assumes a data distribution consistent with a domain decomposition approach, where all variables and equations related to a given portion of the computation domain are assigned to a process; the data distribution can be specified in multiple ways allowing easy interfacing with many graph partitioning procedures. The library design also provides data management tools allowing easy interfacing with data assembly procedures typical of finite elements and finite volumes discretization. Versions of the library have been used in various application domains, mostly in fluid dynamics and structural analysis, where it has been successfully used to solve linear system with millions of unknowns arising in complex simulations. The PSBLAS library version 3.0 is implemented with Fortran 2003. PSBLAS contains 10 sub-packages (*prec, psblas, util, impl, krylov, tools, serial, internals, comm, and modules*), 476 files, and 135 modules.

4.2.4 MLD2P4

Multi-Level Domain Decomposition Parallel Preconditioners Package based on PSBLAS (MLD2P4 Version 1.2) is a package of parallel algebraic multi-level preconditioners. This package provides a variety of high-performance preconditioners for the Krylov methods of PSBLAS. A preconditioner is an operator capable of reducing the number of iterations needed to achieve convergence to the solution of a linear system; multilevel preconditioners are very powerful tools especially suited for problems derived from elliptic PDEs. This package is implemented with object-based Fortran 95. The MLD2P4 contains only one package (*miprec*), 117 files and 9 modules.

4.2.5 MPFlows

Multiphase flows (MPFlows) is a package developed for computational modeling of spray applications. MPFlows is implemented with Fortran 2003/2008. The use of coarrays within this application enables HPC software works without requiring external parallel libraries. MPFlows contains 2 sub-packages (*spray* and *utilities*), 12 files, and 12 modules.

4.3 Analysis

Table 2 shows the results of experiments. Each cell represents the recall as a ratio between extracted data and actual data. The results show that the recall reaches 100% for all sub-packages. Overall, there was only one error in *precision* in the *ForTrilinos* sub-package of *ForTrilinos*. Our analysis of the code identified a conditional preprocessor statement (specified by the `#if` statement) as the source of the problem. *ForUML* currently does not handle preprocessor directives. During the experiments, only 6 files were not parsed (0.89% of all files). The notification messages informed the users which files were not processed and specifically why each file could not be processed. Based on code inspection, we found four files that do not conform to the Fortran model described earlier (Figure 1). Those files do not have the `module` keyword that is the starting point for the transformation process. Other files exceptions were due to ambiguous syntax, e.g. Fortran keywords were used as part of a procedure name (e.g., `print`, `allocate`). Table 2 only shows the results for packages that have the *Type* construct. We only evaluated the correctness of *ForUML* current capabilities.

Figure 5 provides an example of an excerpt (due to space constraints) from a class diagram derived from MPFlows. In Fortran, each dummy argument has three possible intent attributes including *IN*, *OUT*, and *INOUT*. Therefore each parameter, which is passed to the operation in the diagram, needs to be specified with a specific intent. In the class diagram, the keyword *IN* is omitted because ArgoUML assumes that a parameter has the *IN* by default.

Table 3: A brief comparison between UML tools

Features	**Rose Enterprise**[12]	**Doxygen**	**Libthorin**	**ForUML+ArgoUML**	**Rigi**[13]
Visualization	UML	Graph	UML	UML	Graph
Reverse Eng.(Fortran)	No	No	Ver.90	Yes	No
Hide/Show Detail	Yes	No	Yes	No	No
Inheritance	Yes	No	Yes	Yes	No
Layout	A/M	A	A	A/M	A

Note: Automatically adjusted (A) and Manually adjusted (M)

4.4 Discussion

Based on the experimental results, ForUML provided quite precise outputs. *ForUML* was able to automatically transform the source code into the correct UML diagrams. To illustrate the contributions of *ForUML*, Table 3 compares *ForUML* with other visualization-based tools [25] that have features to support program comprehension tasks. Based on this table, one of the unique contributions of *ForUML* is its ability to reverse engineering OO Fortran code. *ForUML* integrates the capabilities of *ArgoUML* to visually display the class diagram. However, *ForUML* has a few limitations that must be addressed in the future:

- Provide more relationship types. One example of other relationship types in UML is *dependency*. In practice, dependency is most commonly used between elements (e.g., packages, folders) that contain other elements located in different packages.

- Incorporation of other UML CASE[10] tools. Currently, *ForUML* integrates *ArgoUML* as the CASE tool. We plan to build different interfaces to integrate with other UML tools, so users can select their tool of preference. Although many UML CASE tools support the use of XMI documents, there are several XMI versions defined by Object Management Group (OMG) and different tools support different versions. We also plan to develop a plugin for Photran[11] (Fortran IDE based on Eclipse), to allow users to automatically generate UML diagrams within the IDE.

[10]Computer-aided Software Engineering
[11]http://www.eclipse.org/photran/
[12]http://www-03.ibm.com/software/products/us/en/enterprise/
[13]http://www.rigi.csc.uvic.ca

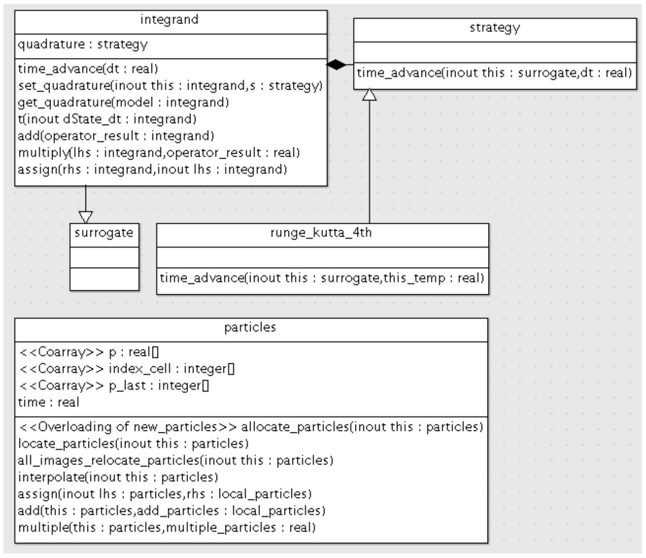

integrand
quadrature : strategy
time_advance(dt : real)
set_quadrature(inout this : integrand,s : strategy)
get_quadrature(model : integrand)
t(inout dState_dt : integrand)
add(operator_result : integrand)
multiply(lhs : integrand,operator_result : real)
assign(rhs : integrand,inout lhs : integrand)
strategy
time_advance(inout this : surrogate,dt : real)
surrogate
runge_kutta_4th
time_advance(inout this : surrogate,this_temp : real)
particles
<<Coarray>> p : real[]
<<Coarray>> index_cell : integer[]
<<Coarray>> p_last : integer[]
time : real
<<Overloading of new_particles>> allocate_particles(inout this : particles)
locate_particles(inout this : particles)
all_images_relocate_particles(inout this : particles)
interpolate(inout this : particles)
assign(inout lhs : particles,rhs : local_particles)
add(this : particles,add_particles : local_particles)
multiple(this : particles,multiple_particles : real)

Figure 5: The Class Diagram (partial) : MPFlows

4.5 Experience

ForUML played a significant role in the development of the CLiiME package [19]. During the development process developers used ForUML to validate the design after each code refactoring process. Developers compared the class diagram obtained from ForUML with the originally agreed upon design. After comparison, they determined instances in which the code implementation deviated from the design. Instead of inspecting the source code manually, developers were able to make the comparison/decision with less effort. Also, during development, developers were able to use the extracted UML diagrams to identify code smells, places where the code might induce some defects in the future. For instance, we inspected the UML class diagrams and identify the places where classes had too many type-bound procedure, procedures with too many arguments, etc. All of which was corrected during the refactoring process. This project also deployed two GoF design patterns [11]. We used *ForUML* to confirm the correct implementation of those two design patterns rather than reviewing the source code. In addition to help developers in the CLiiME project, ForUML also influenced the development of PSBLAS version 3.1, by allowing a comprehensive and unitary view of the project.

The UML diagram must be properly arranged to foment design comprehension. A large class diagram that contains several classes and relationships requires additional effort from users' as the try to assimilate all the information. Unfortunately, the built-in function layout in ArgoUML does not refine the layout in diagrams that contain numerous elements. Although ArgoUML provides the ability to zoom in or zoom out, large diagrams can still be difficult to view. These problems can be addressed by dividing the collection of classes into smaller packages, which should improve the diagram's understandability. Another option is to provide different settings for the information included in the class diagrams, allowing a user to create diagrams with the level of detail required for a particular task. This option can ease the development and/or maintenance process by eliminating irrelevant information.

5. CONCLUSION

This paper presents and evaluates the *ForUML* tool that can be used for extracting UML class diagrams from Fortran code. Fortran is one of the predominant programming languages used in the HPC community. *ForUML* generates a visual representation of software implemented in OO Fortran in the same way as is done in other, more traditional, OO languages. Currently, *ForUML* can produce an XMI document that describes the UML Class Diagrams. The tool supports the inheritance and composition relationships that are the most common relationships found in software systems. The tool integrates ArgoUML, an open source UML modeling tool to allow users to view and modify the UML diagrams without installing a separate UML modeling tool.

We have run *ForUML* on five CSE software packages to generate class diagrams. The experimental results showed that *ForUML* generates highly accurate UML class diagrams. Based on the UML class diagrams generated by *ForUML*, we identified a few limitations of its capabilities. To augment the results of experiments, we have created a website that contains all of the diagrams generated by *ForUML* along with a video demonstrating the use of the tool. We plan to add more diagrams to the website as we run *ForUML* on additional software packages.

We believe that *ForUML* allows developers to extract design diagrams from their source code. These diagrams have proved helpful when developers need to validate whether the code under development conforms to the original design. Similarly, when developers refactor the code, they can ensure that the refactoring process does not break exiting functionality or decompose the architecture. Additionally, *ForUML* generates documentation that describes the high-level structure of the software. This documentation should make communication between developers and the stakeholders or customers more efficient.

In the future, we plan to address the limitations we have identified. We also plan to add features to generate UML sequence diagram. A single class diagram does not sufficiently describe the entire software system. Sequence diagrams are widely used to represent the interactive behavior of the subject system [4]. To create UML sequence diagrams, we would have to augment the *ForUML* Extractor to build the necessary relationships among objects necessary for the Generator to create the corresponding XMI code. Additionally, we will conduct human-based studies to evaluate the effectiveness and usability of *ForUML* by other members of the HPC software developer community. To encourage wider adoption and use of *ForUML*, we are investigating the possibility of releasing it as open source software. This direction can help us to get more feedback about the usability and correctness of the tool. Demonstrating that *ForUML* is a realistic tool for large-scale computational software will make it an even more valuable contribution to both the CSE and HPC communities.

Acknowledgments

The authors gratefully thank Dr. Jeffrey Carver of the University of Alabama, Dr. Damian W. I. Rouson, at Stanford University and Hope A. Michelsen member of the Combustion Chemistry Department at Sandia National Laboratories, their useful comments and helpful discussions were extremely valuable.

6. REFERENCES

[1] M. H. Alalfi, J. R. Cordy, and T. R. Dean. SQL2XMI: Reverse Engineering of UML-ER Diagrams from Relational Database Schemas. In *Proceedings of the 15th Working Conference on Reverse Engineering*, WCRE '08, pages 187–191, Washington, DC, USA, 2008. IEEE Computer Society.

[2] M. H. Alalfi, J. R. Cordy, and T. R. Dean. Automated Reverse Engineering of UML Sequence Diagrams for Dynamic Web Applications. In *Proceedings of the IEEE International Conference on Software Testing, Verification, and Validation Workshops*, ICSTW '09, pages 287–294, Washington, DC, USA, 2009. IEEE Computer Society.

[3] D. Barbieri, V. Cardellini, S. Filippone, and D. Rouson. Design Patterns for Scientific Computations on Sparse Matrices. In *Proceedings of the International Conference on Parallel Processing*, Euro-Par'11, pages 367–376, Berlin, Heidelberg, 2012. Springer-Verlag.

[4] L. C. Briand, Y. Labiche, and Y. Miao. Towards the Reverse Engineering of UML Sequence Diagrams. In *Proceedings of the 10th Working Conference on Reverse Engineering*, WCRE '03, pages 57–66, Washington, DC, USA, 2003. IEEE Computer Society.

[5] J. C. Carver. Software Engineering for Computational Science and Engineering. *Computing in Science Engineering*, 14(2):8–11, 2011.

[6] J. C. Carver, R. P. Kendall, S. E. Squires, and D. E. Post. Software Development Environments for Scientific and Engineering Software: A Series of Case Studies. In *Proceedings of the 29th International Conference on Software Engineering*, ICSE '07, pages 550–559, Washington, DC, USA, 2007. IEEE Computer Society.

[7] V. Decyk, C. Norton, and H. Gardner. Why Fortran? *Computing in Science Engineering*, 9(4):68–71, July-Aug.

[8] V. K. Decyk, C. D. Norton, and B. K. Szymanski. How to Support Inheritance and Run-time Polymorphism in Fortran 90. *Computer Physics Communications*, 115:9–17, 1998.

[9] E. B. Duffy and B. A. Malloy. A Language and Platform-Independent Approach for Reverse Engineering. In *Proceedings of the Third ACIS Int'l Conference on Software Engineering Research, Management and Applications*, SERA '05, pages 415–423, Washington, DC, USA, 2005. IEEE Computer Society.

[10] S. Filippone and A. Buttari. Object-Oriented Techniques for Sparse Matrix, Computations in Fortran 2003. *ACM Transactions on Mathematical Software*, 38(4):1–20, Aug 2012.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[12] E. Gansner, E. Koutsofios, S. North, and K.-P. Vo. A Technique for Drawing Directed Graphs. *IEEE Transactions on Software Engineering*, 19(3):214–230, mar 1993.

[13] E. Korshunova, M. Petkovic, M. van den Brand, and M. Mousavi. CPP2XMI: Reverse Engineering of UML Class, Sequence, and Activity Diagrams from C++ Source Code. In *13th Working Conference on Reverse Engineering, 2006. WCRE '06.*, pages 297 –298, oct. 2006.

[14] T. C. Lethbridge, S. Tichelaar, and E. Ploedereder. The dagstuhl middle metamodel: A schema for reverse engineering. *Electronic Notes in Theoretical Computer Science*, 94(0):7 – 18, 2004.

[15] E. Loh. The ideal hpc programming language. *Queue*, 8(6):30:30–30:38, June 2010.

[16] T. Mens and T. Tourwe. A survey of software refactoring. *IEEE Transactions on Software Engineering*, 30(2):126–139, 2004.

[17] Z. Merali. Computational Science: ...Error. *Nature*, 467(7317):775–777, Oct. 2010.

[18] K. Morris, D. W. Rouson, M. N. Lemaster, and S. Filippone. Exploring Capabilities within ForTrilinos by Solving the 3D Burgers Equation. *Sci. Program.*, 20(3):275–292, Oct. 2012.

[19] A. Nanthaamornphong, K. Morris, D. Rouson, and H. Michelsen. A Case Study: Agile Development in the Community Laser-Induced Incandescence Modeling Environment (CLiiME). In *International Workshop on Software Engineering for Computational Science and Engineering (In Conjunction with ICSE 2013)*, accepted Feb, 2013.

[20] J. Overbey, S. Xanthos, R. Johnson, and B. Foote. Refactorings for fortran and high-performance computing. In *Proceedings of the second international workshop on Software engineering for high performance computing system applications*, SE-HPCS '05, pages 37–39, New York, NY, USA, 2005. ACM.

[21] M. Pacione. Software Visualization for Object-Oriented Program Comprehension. In *Proceedings of the IEEE 26th International Conference on Software Engineering*, pages 63–65, May.

[22] D. W. Rouson, J. Xia, and X. Xu. Object Construction and Destruction Design Patterns in Fortran 2003. *Procedia Computer Science*, 1(1):1495 – 1504, 2010.

[23] D. W. I. Rouson, H. Adalsteinsson, and J. Xia. Design Patterns for Multiphysics Modeling in Fortran 2003 and C++. *ACM Trans. Math. Softw.*, 37(1):3:1–3:30, Jan. 2010.

[24] J. Segal. Professional End User Developers and Software Development Knowledge. Technical Report 2004/25, Open University UK, 2004.

[25] M.-A. Storey. Theories, Tools and Research Methods in Program Comprehension: Past, Present and Future. *Software Quality Control*, 14(3):187–208, Sept. 2006.

[26] P. Tonella and A. Potrich. Reverse Engineering of the UML Class Diagram from C++ Code in Presence of Weakly Typed Containers. In *Proceedings of the IEEE International Conference on Software Maintenance*, ICSM '01, pages 376–385, Washington, DC, USA, 2001. IEEE Computer Society.