Hindawi Publishing Corporation
Scientific Programming
Volume 2015, Article ID 421816, 15 pages
http://dx.doi.org/10.1155/2015/421816

Research Article

Extracting UML Class Diagrams from Object-Oriented Fortran: ForUML

Aziz Nanthaamornphong,[1] Jeffrey Carver,[2] Karla Morris,[3] and Salvatore Filippone[4]

[1] *Department of Information and Communication Technology, Prince of Songkla University, Phuket Campus, Phuket 83120, Thailand*
[2] *Department of Computer Science, University of Alabama, Tuscaloosa, AL 35487, USA*
[3] *Sandia National Laboratories, 7011 East Avenue, Livermore, CA 94550-9610, USA*
[4] *Department of Civil and Computer Engineering, University of Rome 'Tor Vergata', Roma 00173, Italy*

Correspondence should be addressed to Aziz Nanthaamornphong; aziz.nantha@gmail.com

Received 10 April 2014; Accepted 20 June 2014

Academic Editor: Selim Ciraci

Many scientists who implement computational science and engineering 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 OO 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 evolution. The experiments carried out to evaluate the proposed tool show its accuracy and a few of the limitations.

1. Introduction

Computational research has been referred to as the third pillar of scientific and engineering research, along with experimental and theoretical research [1]. Computational science and engineering (CSE) researchers develop software to simulate natural phenomena that cannot be studied experimentally or to process large amounts of data. CSE software has a large impact on society as it is used by researchers to study critical problems in a number of important application domains, including weather forecasting, astrophysics, construction of new physical materials, and cancer research [2]. For example, US capabilities in science and engineering are frequently called upon to address urgent challenges in national and homeland security, economic competitiveness, health care, and environmental protection [3]. Recently the software engineering (SE) community has become more interested in the development of software for CSE research.

In this critical type of software, Fortran is still a very widely used programming language [4]. 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 required by CSE software. In the case of Fortran developers, this OOP paradigm was first emulated following an object-based approach in Fortran 90/95 [5–7]. By including full support for OOP constructs, the Fortran 2003 language standard influenced the advent of several CSE packages [8–12].

One of the greatest challenges faced by CSE developers is the ability to effectively maintain their software over its generally long lifetime [13]. This challenge implies high development and maintenance costs during a software system's lifetime. The difficulty of the maintenance process is affected by at least three factors. First, most CSE developers are not formally trained in SE. Second, some existing SE tools are difficult to use in CSE development. In general, CSE developers request tools to accommodate documentation, correctness testing, and aid in design software for testability. Unfortunately, most SE tools were not designed to be used in the context of CSE development. Third, CSE software often lacks the formal documentation necessary to help developers understand its complex design. This lack of documentation presents an even larger software maintenance challenge. The objective of this work is to provide tool support for automatically extracting UML class diagrams from OO Fortran code.

To address this objective, we developed and evaluated the *ForUML* tool. *ForUML* uses a reverse engineering approach to transform Fortran source code into UML models. To ensure flexibility, our solution uses a Fortran parser that does not depend on any specific Fortran compiler and generates output in the XML Metadata Interchange (XMI) format. The tool then displays the results of the analysis (the UML class diagram) using the *ArgoUML* (http://argouml.tigris.org/) modeling tool. We evaluated the accuracy of *ForUML* using five CSE software packages that use object-oriented features from the Fortran 95, 2003, and 2008 compiler standards. This paper extends the workshop paper [14] by providing more background information and more details on the transformation process in ForUML. Additionally, this paper includes a discussion of the audience feedback during the *Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering* (SE-HPCCSE'13).

The contributions of this paper are as follows:

(i) the ForUML tool that will help CSE developers extract UML design diagrams from OO Fortran code to enable them make good decisions about software development and maintenance tasks;

(ii) description of the transformation process used to develop ForUML, which may help other tool authors create tools for the CSE community;

(iii) the results of the evaluation and our experiences using ForUML on real CSE projects to highlight its benefits and limitations;

(iv) workshop feedback that should help SE develop practices and tools that are suitable for use in the CSE domain.

The rest of this paper is organized as follows. Section 2 provides the background concepts related to this work. Section 3 presents ForUML. Section 4 describes the evaluation and our experiences with ForUML. Section 5 discusses the evaluation results and limitations of ForUML. Finally, Section 6 draws conclusions and presents future work.

2. Related Work

This section first describes important CSE characteristics that impact the development of tool support. Next, it presents two important concepts used in the development of ForUML, reverse engineering and OO Fortran. Finally, because one of the benefits of using ForUML is the ability to recognize and maintain design patterns, the last subsection provides some background on design patterns.

2.1. Important CSE Characteristics. This section highlights three characteristics of CSE software development that differentiate it from traditional software development. First, CSE developers typically have a strong background in the theoretical science but often do not have formal training about SE techniques that have proved successful in other software areas. More specifically, because the complexity of the problems addressed by CSE generally requires a domain expert (e.g., a Ph.D. in physics or biology) to even understand the problem, and that domain expert generally must learn how to develop software [15]. Wilson [16] stated that one of the reasons why scientists tend to be less effective programmers is that they do not have the time to learn yet another programming language and software tool. Furthermore, the CSE culture, including most funding agencies, tends to view software as the means to a new scientific discovery rather than as a CSE instrument that must be carefully engineered, maintained, and extended to enable novel science.

Second, some SE tools are difficult to use in a CSE development environment [17]. CSE applications are generally developed with software tools that are crude compared to those used today in the commercial sector. Researchers and scientists seek easy-to-use software that enables analysis of complex data and visualization of complicated interactions. 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. Scientists interested in scientific research cannot spend most of their time understanding and using complex software tools. The limited interoperability of the tools and their complexity are major obstructions to their adaptation by the CSE community. For example, Storey noted that CSE developers who lack formal SE training need help with program comprehension when they are developing complex applications [18]. To address this problem, the SE community must develop tools that satisfy the needs of CSE developers. These tools must allow the developers to easily perform important reverse engineering tasks. More specifically, a visualization-based tool is appropriate for program comprehension in complex object-oriented applications [19].

Third, CSE software typically lacks adequate development-oriented documentation [20]. 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 developers of related systems cannot correctly understand how to interact with the subject system.

In addition, the lack of documentation makes refactoring and maintenance difficult and error prone. CSE software typically evolves over many years and involves multiple developers [21], as functionality and capabilities are added or extended [22]. The developers need to be able to determine whether the evolved software deviates from the original design intent. To ease this process, developers need tools that help them identify changes that affect the design and determine whether those changes have undesired effects on design integrity. The availability of appropriate design documentation can reduce the likelihood of poor choices during the maintenance process.

2.2. Reverse Engineering. Reverse engineering is a method that transforms source code into a model [23]. ForUML builds upon and expands some existing reverse engineering work. The Dagstuhl middle metamodel(DMM) is a schema for describing the static structure of source code [24]. DMM supports reverse engineering by representing models extracted from source code written in most common OOP languages. We applied the idea of DMM to OO Fortran.

The transformation process in ForUML is based on the XMI format, which provides a standard method of mapping an object model into XML. XMI is an open standard that allows developers and software vendors to create, read, manage, and generate XMI tools. Transforming the model (Fortran code) to XMI requires use of the model driven architecture (MDA) technology [25], a modeling standard developed by the object management group (OMG) [26]. MDA aims to increase productivity and reuse by using separation of concerns and abstraction. A platform independent model (PIM) is an abstract model that contains the information to drive one or more platform specific models (PSMs), including source code, data definition language (DDL), XML, and other outputs specific to the target platform. MDA defines transformations that map from PIMs to PSMs.

The basic idea of using an XMI file to maintain the metadata for UML diagrams was drawn from four reverse engineering tools. Alalfi 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 [27] and a tool to create UML-entity relationship diagrams for the structured query language (SQL) [28]. Similarly, Korshunova et al. [29] 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) [30]. Duffy and Malloy [31] 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 (http://www.dwarfstd.org/), which is a debugging file format 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 compiler independent and able to generate UML for all types of 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 limitations. First, it does not support all OOP features within Fortran (e.g., type-bound procedures and components). 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 cannot 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 takes advantages of this feature to display the UML diagrams described by the XMI files it generates 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 engineering 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, and operator overloading.

2.3. Object-Oriented Fortran. Fortran is an imperative programming language. Traditionally, Fortran code has been developed through a procedural programming approach that emphasizes the procedures and subroutines in a program rather than the data. A number of studies discuss approaches for expressing OOP principles in Fortran 90/95. For example, Decyk described how to express the concepts of data encapsulation, function overloading, classes, objects, and inheritance in Fortran 90 [6, 7, 32]. Moreover, several authors have described the use and syntax of OO features in Fortran 2003 [33–35]. Table 1 presents important Fortran-specific terms along with their OOP equivalent and some examples of Fortran keywords.

The Fortran 2003 compiler standard added support for OOP, including the following OOP principles: dynamic and static polymorphism, inheritance, data abstraction, and encapsulation. Currently, a number of Fortran compiler vendors support all (or almost all) of the OOP features included in the Fortran 2003 standard. These compilers include [36]

(i) NAG (http://www.nag.com/);

(ii) GNU Fortran (http://gcc.gnu.org/fortran/);

(iii) IBM XL Fortran (http://www-142.ibm.com/software/products/us/en/fortcompfami/);

Table 1: Object-oriented Fortran terms (adapted from [12]).

Fortran	OOP equivalent	Fortran keywords
Module	Package	Module
Derived type	Abstract data type (ADT)	Type
Component	Attribute	—
Type-bound procedure	Method	Procedure
Parent type	Parent class	—
Extend type	Child class	Extends
Intrinsic type	Primitive type	For example, real, integer

```
(1) module example
(2)    type shape_
(3)      real :: area
(4)      integer  :: m[*]
(5)    end type
(6)    !  Inheritance
(7)    type, extends (shape_)  :: circle
(8)      !  Data abstraction
(9)      private
(10)     !  Encapsulation
(11)     real  :: radius
(12)   contains
(13)     procedure  :: set_radius
(14)     procedure  :: add
(15)     procedure   :: area
(16)     !  Polymorphism
(17)     generic  :: total => area
(18)     generic  :: operator(+) => add
(19)   end type
(20)   !  Overloads intrinsic constructor
(21)   interface circle
(22)     module procedure new_circle
(23)   end interface
(24)   !  ...
(25) end module
```

Algorithm 1: Samples code snippet of OOP constructs supported by Fortran 2003.

(iv) Cray (http://www.nersc.gov/users/software/compilers/cray-compilers/);

(v) Intel Fortran (https://software.intel.com/en-us/fortran-compilers).

Fortran 2003 supports procedure overriding where developers can specify a type-bound procedure in a child type that has the same binding name as a type-bound procedure in the parent type. Fortran 2003 also supports user-defined constructors that can be implemented by overloading the intrinsic constructors provided by the compiler. The user-defined constructor is created by defining a generic interface with the same name as the derived type.

Algorithm 1 illustrates a snippet of Fortran 2003 code in which the parent type `shape_` (Line 2) is extended by the type `circle` (Line 7). At runtime the compiler invokes the type-bound procedure `add` (Line 18) whenever an operator "+" (with the specified argument type) is used in the client code. This behavior conforms to polymorphism, which allows a type or procedure to take many object or procedural forms.

Data abstraction is the separation between the interface and implementation of the program. It allows developers to provide essential information about the program to the outside world. In Fortran, the `private` and `public` keywords control access to members of the type. Members defined with `public` are accessible to any part of the program. Conversely, members defined with `private` are not accessible to code outside the module in which the type is defined. In the example, the component `radius` (Line 11) cannot be accessed directly by other programs. Rather, the caller must invoke the type-bound procedure `set_radius` (Line 13).

With the increase in parallel computing, the CSE community needs to utilize the full processing power of all available resources. Fortran 2008 improves the performance for a parallel processing feature by introducing the Coarray model [37]. The Coarray extension allows developers to express data distribution by specifying the relationship between memory images/cores. The syntax of the Coarray is very much like normal Fortran array syntax, except with square brackets instead of parentheses. For example, the statement `integer:: m[*]` (Line 4) declares `m` to be an integer that is sharable across images. Fortran uses normal rounded brackets () to point to data in local memory. Although using Coarray requires the additional syntax, the coarray has been designed to be easy to implement and to provide the compiler scope both to apply its optimizations within each image and among images.

2.4. Design Patterns. A design pattern is a generic solution to a common software design problem that can be reused in similar situations. Design patterns are made of the best practices drawn from various sources, such as building software applications, developer experiences, and empirical studies. Generally, we can classify the design patterns of the software into classical and novel design patterns. The 23 classical design patterns were introduced by the "Gang of Four" (GoF) [38]. Subsequently, software developers and researchers have proposed a number of novel design patterns targeted at particular domains, for example, parallel programming [39, 40].

In general, a design pattern includes a section known as *intent*. Intent is "a short statement that answers the following questions: What does the design pattern do? What is its rationale and intent? What particular design issues or problem does it address?" [38]. For example, the intent of the template method pattern requires that developers define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure. When using design patterns, developers have to understand the intent of each design pattern to determine whether the design pattern could provide a good solution to a given problem.

Several researchers have proposed design patterns for computational software implemented with Fortran. For example, Weidmann [41] implemented design patterns to enable sparse matrix computations on NVIDIA GPUs. They then evaluated the benefits of the implementation and reported that the design patterns provided a high level of maintainability and performance. Rouson et al. [12] proposed three new design patterns, called multiphysics design patterns, to implement the differential equations, which are integrated into multiphysics and numerical software. These new design patterns include the semidiscrete, surrogate and template class patterns. Markus demonstrated how some well-known design patterns could be implemented in Fortran 90, 95, and 2003 [42, 43]. Similarly, Decyk et al. [4] proposed the factory pattern in Fortran 95 based on CSE software. These researchers presented the proposed pattern implementation in their particle-in-cell (PIC) methods [44] in plasma simulation software. Decyk and Gardner [45] also described a way to implement the strategy, template, abstract factory, and facade patterns in Fortran 90/95.

3. ForUML

This section describes the rationale and benefits of developing ForUML and details the transformation process used by ForUML.

3.1. The Need for ForUML. The CSE characteristics described in Section 2.1 indicate that CSE developers could benefit from a tool that creates system documentation with little effort. The SE community typically uses reverse engineering to address this problem.

Although there are a number of reverse engineering tools [46] (see Section 2.2), those tools that can be applied to OO Fortran do not provide the full set of documentation required by developers. Therefore, we identified the need for a tool that automatically reverses engineers OO Fortran code into the necessary UML design documentation.

This work is primarily targeted at CSE developers who develop OO Fortran. The ForUML tool will provide the following benefits to the CSE community.

(1) The extracted UML class diagrams should support software maintenance and evolution and help maintainers ensure that the original design intentions are satisfied.

(2) The developers can use the UML diagrams to illustrate software design concepts to their team members. In addition, UML diagrams can help developers visually examine relationships among objects to identify code smells [47] in software being developed.

(3) Because SE tools generally improve productivity, ForUML can reduce the training time and learning curve required for applying SE practices in CSE software development. For instance, ForUML will help developers perform refactoring activities by allowing them to evaluate the results of refactoring using the UML diagrams rather than inspecting the code manually.

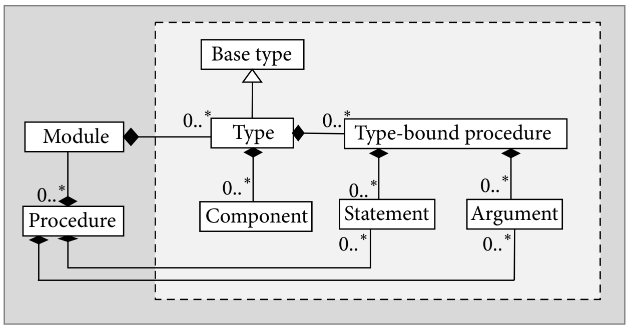


Figure 1: The Fortran model.

Since Fortran 2003 provides all of the concepts of OOP, tools like ForUML can help to place Fortran and other OOP program languages on equal levels.

3.2. Transformation Process. The primary goal of ForUML is to reverse engineering 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, that is, 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 box 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.

3.2.1. Parsing. The Fortran code is parsed by the Open Fortran Parser (OFP) (http://fortran-parser.sourceforge.net/). OFP provides ANTLR-based parsing tools [48], including Fortran grammars and libraries for performing translation actions. ANTLR is a parser generator that can parse language specifications in an EBNF-like syntax, a notation for formally describing programming language syntax, and generate the library to parse the specified language. ANTLR distinguishes three compilation phases: lexical analysis, parsing, and tree walking.

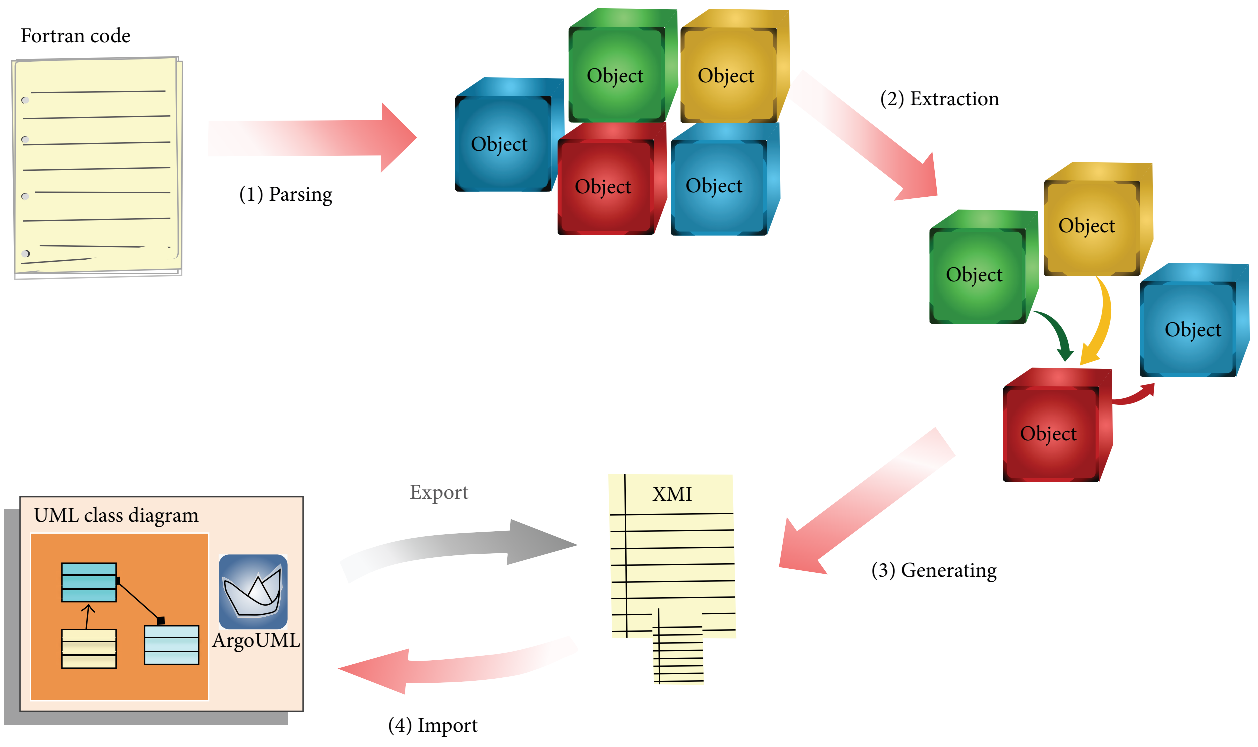


FIGURE 2: The transformation process.

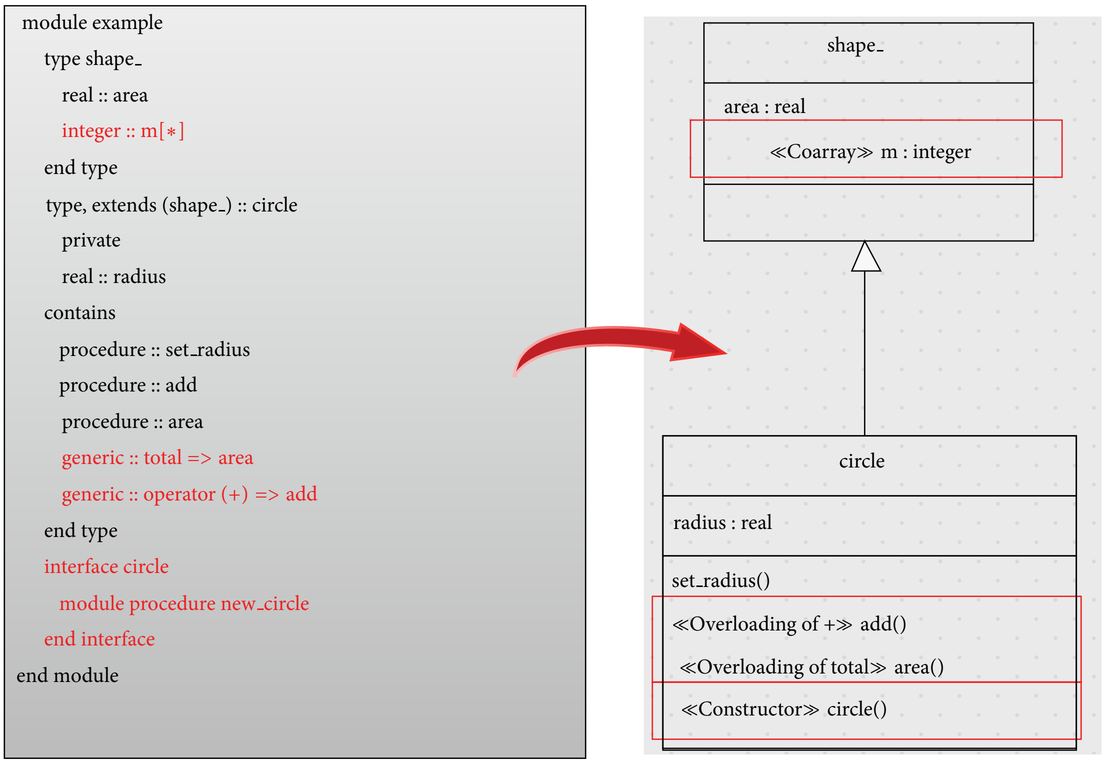
We have customized the ANTLR libraries to translate particular AST nodes (i.e., type, component, and type-bound procedure) into objects. These AST nodes are only the basic elements of UML class diagrams. In fact, a UML class diagram includes classes, attributes, methods, and relations. The parsing actions include two steps. The first step verifies the syntax in the source file and eliminates source files that have syntax problems. It also eliminates source files that do not contain any instances of type and module. For example, ForUML will eliminate modules that contain only subroutines or functions. After this step, ForUML reports the results to the user via a GUI. In the second step, the parser manipulates all AST nodes, relying on the model described earlier. Note that ForUML only manipulates 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.

3.2.2. Extraction. During the extraction process, ForUML excerpts the objects and identifies their relationships. ForUML determines the type of each extracted relationship and maps each relationship to a specific relationship's type object. Based on the example code in Algorithm 1, the type `circle` inherits the type `shape`. As a consequence, the extraction process will create a generalization object. ForUML supports two relationship types: composition and generalization.

(i) Composition represents the whole-part relationship. The lifetime of the part classifier depends on 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. The association refers to a type not provided by the user and as a result it does not appear in the class diagram. In the UML class diagram, a composition relationship appears as a solid line with a filled diamond at the association end that is connected to the whole class.

(ii) 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.

3.2.3. Generating. We developed the XMI generator module to convert the extracted objects into XMI notation based on 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 2. In addition to these rules, we developed new stereotype notations for the constructor, coarray constructs, type-bound procedure overloading, and operator overloading, such as `≪Constructor≫`, `≪Coarray≫`, `≪Overloading≫`, and `≪Overloading of +≫`.



```
module example
  type shape_
    real :: area
    integer :: m[*]
  end type
  type, extends (shape_) :: circle
    private
    real :: radius
  contains
    procedure :: set_radius
    procedure :: add
    procedure :: area
    generic :: total => area
    generic :: operator (+) => add
  end type
  interface circle
    module procedure new_circle
  end interface
end module
```

Figure 3: Sample code snippet of Fortran supported by ForUML.

Table 2: 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")

Figure 3 provides the sample Fortran code without procedure implementation and its generated class diagram including stereotypes.

3.2.4. 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 ("what you see is what you get") 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 that (1) it has seamless integration properties as an open source and Java implementation; (2) it has sufficient documentation; and (3) it 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.

(1) Select the Fortran source code

(2) Select the location to save the output.

(3) Open the UML diagram.

Figures 4–7 show screenshots from the ForUML tool. Figure 4 presents the graphical user interface (GUI) of

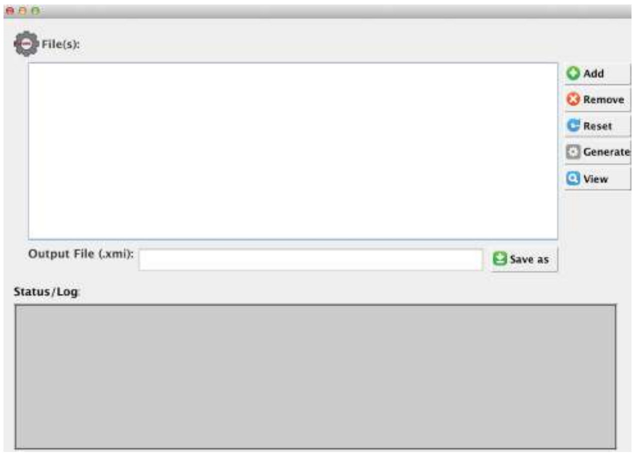
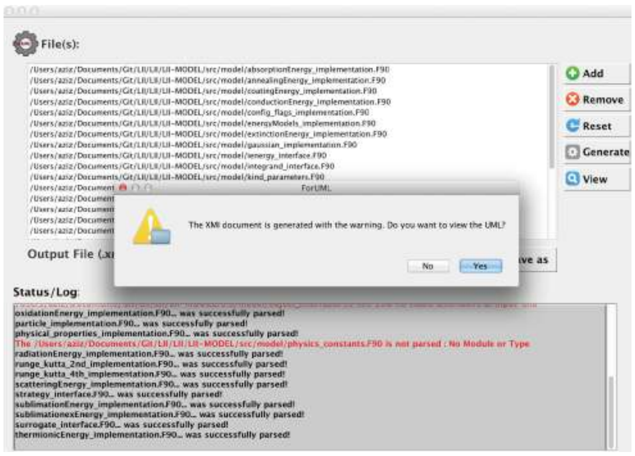


Figure 4: A graphical user interface of ForUML.

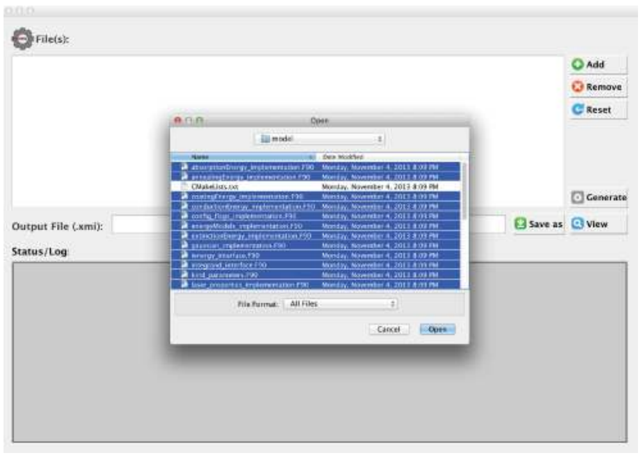


```
/Users/aziz/Documents/Git/LII/LII/LII-MODEL/src/model/absorptionEnergy_implementation.F90
/Users/aziz/Documents/Git/LII/LII/LII-MODEL/src/model/annealingEnergy_implementation.F90
/Users/aziz/Documents/Git/LII/LII/LII-MODEL/src/model/coatingEnergy_implementation.F90
/Users/aziz/Documents/Git/LII/LII/LII-MODEL/src/model/conductionEnergy_implementation.F90
/Users/aziz/Documents/Git/LII/LII/LII-MODEL/src/model/config_flags_implementation.F90
/Users/aziz/Documents/Git/LII/LII/LII-MODEL/src/model/energyModels_implementation.F90
/Users/aziz/Documents/Git/LII/LII/LII-MODEL/src/model/extinctionEnergy_implementation.F90
/Users/aziz/Documents/Git/LII/LII/LII-MODEL/src/model/gaussian_implementation.F90
/Users/aziz/Documents/Git/LII/LII/LII-MODEL/src/model/ienergy_interface.F90
/Users/aziz/Documents/Git/LII/LII/LII-MODEL/src/model/integrand_interface.F90
/Users/aziz/Documents/Git/LII/LII/LII-MODEL/src/model/kind_parameters.F90
/Users/aziz/Document
/Users/aziz/Document
/Users/aziz/Document
/Users/aziz/Document
/Users/aziz/Document
```

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

```
oxidationEnergy_implementation.F90... was successfully parsed!
particle_implementation.F90... was successfully parsed!
physical_properties_implementation.F90... was successfully parsed!
The /Users/aziz/Documents/Git/LII/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!
```

Figure 6: Generating the XMI.



Name	Date Modified
absorptionEnergy_implementation.F90	Monday, November 4, 2013 8:09 PM
annealingEnergy_implementation.F90	Monday, November 4, 2013 8:09 PM
CMakeLists.txt	Monday, November 4, 2013 8:09 PM
coatingEnergy_implementation.F90	Monday, November 4, 2013 8:09 PM
conductionEnergy_implementation.F90	Monday, November 4, 2013 8:09 PM
config_flags_implementation.F90	Monday, November 4, 2013 8:09 PM
energyModels_implementation.F90	Monday, November 4, 2013 8:09 PM
extinctionEnergy_implementation.F90	Monday, November 4, 2013 8:09 PM
gaussian_implementation.F90	Monday, November 4, 2013 8:09 PM
ienergy_interface.F90	Monday, November 4, 2013 8:09 PM
integrand_interface.F90	Monday, November 4, 2013 8:09 PM
kind_parameters.F90	Monday, November 4, 2013 8:09 PM
laser_properties_implementation.F90	Monday, November 4, 2013 8:09 PM

Figure 5: Selection of the Fortran code.

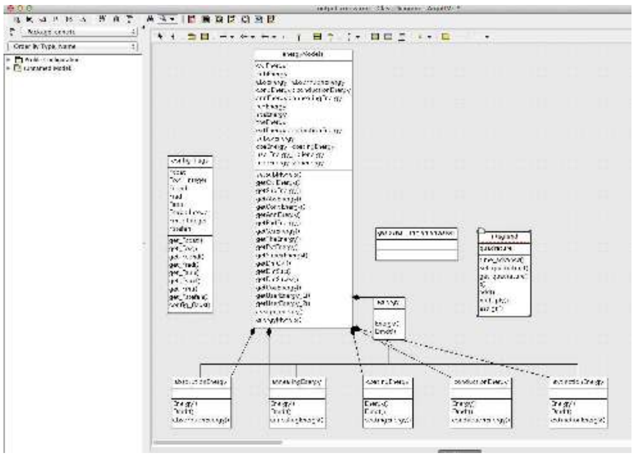


Figure 7: Viewing the UML class diagram.

ForUML. Figure 5 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 6). Once the XMI document is successfully generated, the user can view the class diagram by selecting the *View* button. Figure 7 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

To assess the effectiveness of ForUML, we conducted some small experiments to gather data about its accuracy in extracting UML constructs from code. This section also provides some lessons learned from the studies and feedback from the SEC-HPC'13 workshop audience.

4.1. Controlled Experiment. The following subsections provide the details of a controlled experiment to evaluate ForUML. The accompanying website (http://aziz.students.cs.ua.edu/foruml-eval.htm) contains all of the class diagrams. The website also provides the ForUML executable (source code is not available yet) for download.

4.1.1. 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 and Potrich [49].

(i) *Recall* measures the percentage of the various objects, that is, type, components, type-bound procedure, and associations, in the source code correctly identified by ForUML.

(ii) *Precision* measures the percentage of the objects identified by ForUML that are correct when compared with the source code.

We performed the evaluations as follows.

(1) We manually inspected the source code to document the number of relevant objects in each package. Note that we performed this step multiple times to ensure that the numbers were not biased by human error.

(2) We ran ForUML on each software package and documented the number of relevant objects included in the generated class diagram.

(3) 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) 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).

(5) We investigated whether the generated class diagrams could present the design pattern classes existing in the subject systems.

4.1.2. Subject Systems. The five software packages we used in the experiments were (1) ForTrilinos (http://trilinos.sandia.gov/packages/fortrilinos/); (2) CLiiME; (3) PSBLAS (http://www.ce.uniroma2.it/psblas/); (4) MLD2P4 (http://www.mld2p4.it/); and (5) MPFlows. We selected these software packages because they were intentionally developed for use in the CSE environment. Two of the software packages (CLiiME and MPFlows) are not yet publicly available. A description of each software package follows.

(1) ForTrilinos: ForTrilinos consists of an OO Fortran interface to expand the use of Trilinos (http://trilinos.sandia.gov/) 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 subpackages (epetra, aztecoo, amesos, and fortrilinos), 36 files, and 36 modules.

(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 subpackages (model and utilities), 30 files, and 29 modules. Additionally, this application contains three design patterns, including factory method, strategy, and surrogate.

(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. Researchers have used versions of the library 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 subpackages (prec, psblas, util, impl, krylov, tools, serial, internals, comm, and modules), 476 files, and 135 modules.

(4) *MLD2P4*: multi-level domain decomposition parallel preconditioners package based on PSBLAS (MLD2P4 version 1.2) is a package of parallel algebraic multilevel 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.

(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 scalable CSE software package that works without requiring the use of external parallel libraries. MPFlows contains 2 subpackages (spray and utilities), 12 files, and 12 modules. Note that this package contains two design patterns, including strategy and surrogate.

4.1.3. Analysis. Table 3 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 subpackages. Overall, there was only one error in *precision* in the *ForTrilinos* subpackage 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; for example, Fortran keywords were used as part of a procedure name (e.g., `print`, `allocate`). Table 3 only shows the results for packages that have the

Table 3: Evaluation of ForUML: recall (extracted data/actual data).

Packages	Subpackages	Type	Procedure	Component	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%)

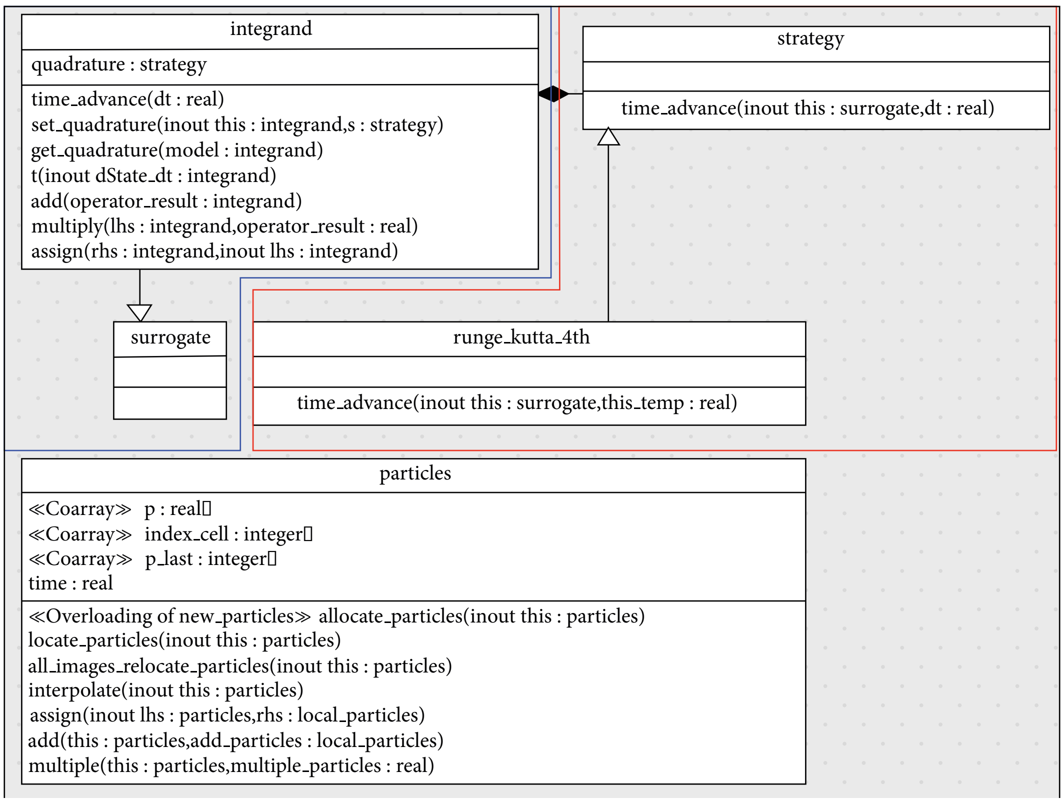

integrand
quadrature : strategy
time_advance(dt : real)
set_quadrature(inout this : integrand,s : strategy)
get_quadrature(model : integrand)
t(inout dState_dt : integrand)
add(operator_result : integrand)
multiply(lhs : integrand,operator_result : real)
assign(rhs : integrand,inout lhs : integrand)
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 8: The class diagram (partial): MPFlows.

type construct. We only evaluated the correctness of ForUML current capabilities.

Figure 8 provides an example of an excerpt from a class diagram derived from MPFlows. This diagram consists of the implementation of two design patterns, including strategy (inside the red box) and surrogate (inside the blue box) patterns. In the strategy pattern, an interface class `strategy` defines only the time integration method, deferring to subclasses the implementation of the actual quadrature schemes. The concrete strategy class (derived class) `runge_kutta_4th` provides the algorithm that presents a part of the `time_advance` method declared by the `strategy` interface. Next, the surrogate pattern is very similar in concept to an ATM. An ATM holds a surrogate database for bank information that exists in another place. The bank's customer can perform transactions through the ATM and circumvent a visit to the bank. The implementation of the surrogate pattern introduces the `surrogate` abstract class (virtual class in C++). The class `integrand` has a component of class `strategy` meaning that

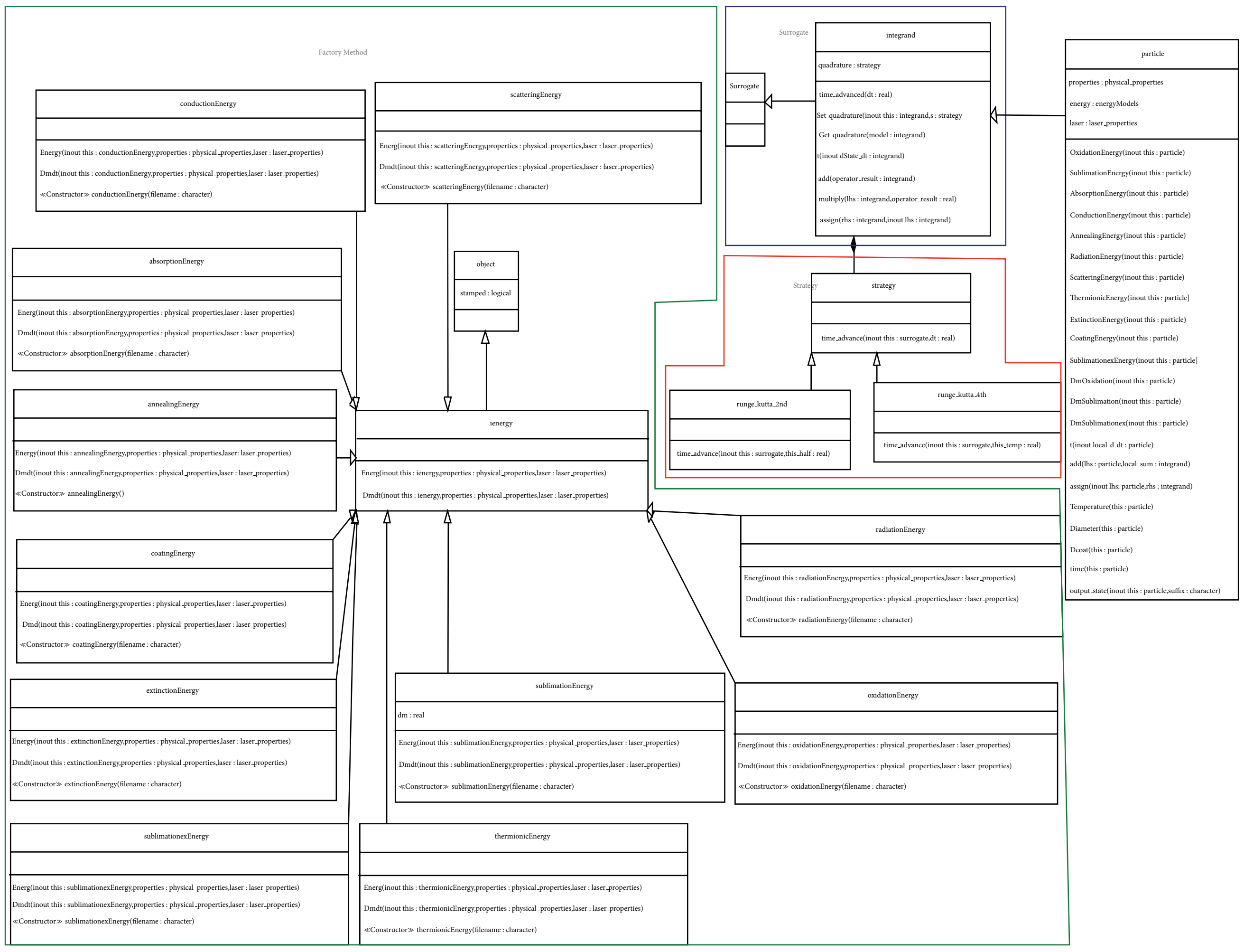


Figure 9: The class diagram (partial): CLiiME.

the `surrogate` allows us to pass an `integrand` child class dummy argument to the type-bound procedures implemented in `runge_kutta_4th`. The class `particle` contains components and type-bound procedures computing the energy of the particle. 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.

4.2. Experience. The following subsections describe our experiences using ForUML on a real CSE project and discuss feedback on ForUML we received during the SE-HPCCSE'13 workshop.

4.2.1. CLiiME Project. ForUML played a significant role in the development of the CLiiME package [50]. The developers used ForUML to validate the design after each code refactoring process. The developers compared the class diagram produced by 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, the developers were able to make the comparison/decision with less effort. Also, the 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 identified places where classes had too many type-bound procedure or procedures with too many arguments, all of which we corrected during the refactoring process.

This project also deployed three design patterns. Figure 9 presents the UML class diagram of the CLiiME project, including the strategy (inside the red box), surrogate (inside the blue box), and factory method (inside the green box) patterns. The factory method pattern indicates encapsulating the subclass selection (∗Energy class) and object construction processes into one class (ienergy). We used ForUML to confirm the correct implementation of those three design patterns rather than reviewing the source code. In addition to helping CLiiME developers, ForUML also influenced the

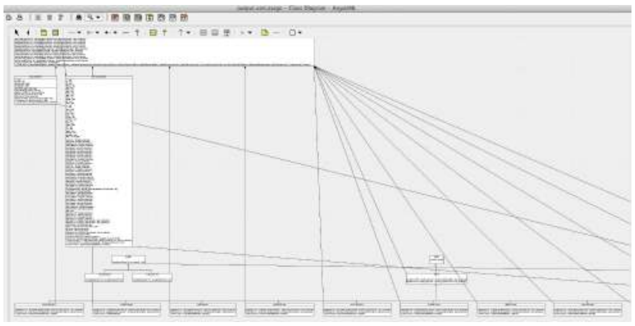


Figure 10: Example of larger classes.

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 they 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. Figure 10 shows an example of a UML class diagram generated by ForUML that includes large classes. 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.

4.2.2. SEC-HPC'13 Workshop. In addition to our own experiences, we can make some observations based on the discussions during the SE-HPCCSE'13 workshop regarding the use of UML in CSE applications. UML helps partition the coding workloads in large projects. For larger projects, especially libraries, it is a matter of dwelling on the "use cases" and designing an interface perhaps with UML. Then feature coding tasks can be distributed to other developers. In contrast, CSE has been reluctant to adopt object-oriented design, whereas in other standard mathematics, linear algebra design bears some similarly to OOP considering larger mathematical structures as objects. Many audiences believed that better SE practices, including adoption ForUML could lead to a better adaptation of codes to multiple architectures. However, one reason for the lack of advance SE in CSE is that CSE developers try to use UML for everything. The audience suggested that other domain specific languages (DSLs) could be useful targets for generating information from legacy code. Further, during the workshop's discussion, there were some questions that inspired us to study the impact of ForUML on the CSE community. We believe that we can find answers to these questions by conducting human-based studies of ForUML. Below is a list of questions that arose during the workshop.

(i) Is UML really useful for CSE developers?

(ii) Can ForUML and UML support larger application sizes and multiple developers?

(iii) Many graphical design models serve multiple purposes. Some users can convey a high-level design for discussion, and others want to display the low-level of design. In the context of CSE software development, does UML serve all these needs well?

(iv) Which aspects of the CSE application should be documented in the UML?

5. Discussion

Based on the experimental results, ForUML provided quite precise outputs. ForUML was able to automatically transform the source code into correct UML diagrams. To illustrate the contributions of ForUML, Table 4 compares ForUML with other visualization-based tools [18] that have features to support program comprehension tasks. Based on this table, one of the unique contributions of ForUML is its ability to reverse engineered OO Fortran code. ForUML integrates the capabilities of ArgoUML to visually display the class diagram.

We believe that ForUML can be used by three types of people during the software development process, especially for CSE software.

(1) Stakeholders or customers: 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.

(2) Developers: ForUML helps developers extract design diagrams from their code. Developers might need to validate whether the code under development conforms to the original design. Similarly, when developers refactor the code, they need to ensure that the refactoring does not break exiting functionality or decompose the architecture.

(3) Maintainers: they need a document that provides adequate design information to enable them to make good decisions. In particular, maintainers who are familiar with other OOP languages can understand a system implemented with OO Fortran.

However, ForUML has a few limitations that must be addressed in the future as follows.

(1) Provide more relationships: two other relationships that we frequently found in the Fortran applications are as follows.

 (i) Dependency: in practice, dependency is most commonly used between elements (e.g., packages, folders) that contain other elements located in different packages. The relationship

Table 4: A brief comparison between UML tools.

Features	Rose enterprise [53]	Doxygen	Libthorin	ForUML + ArgoUML	Rigi [54]
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).

is represented by a dashed line with an arrow pointing toward a class that is an argument in a procedure that is bound to another class.

(ii) Realization: it refers to the links between either the interface or abstract and its implementing classes. A dashed line is connected to an open triangle for a type that extends an abstract type.

Note that although the current version of ForUML does not support these relation types, the users can edit the relationships in the ArgroUML after importing the XMI document.

(2) Incorporation of other UML visualization 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 (http://www.eclipse.org/photran/), to allow users to automatically generate UML diagrams within the IDE.

(3) Generate UML sequence diagram: a single diagram does not sufficiently describe the entire software system. Sequence diagrams are widely used to represent the interactive behavior of the subject system [51]. 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.

6. Conclusion and Future Work

This paper presents and evaluates the ForUML tool that can be used for extracting UML class diagram from Fortran code. Fortran is one of the predominant programming languages used in the CSE software domain. ForUML generates a visual representation of software implemented in OO Fortran in the same way as is done in other, more traditional OO languages. Software developers and practitioners can use ForUML to improve the program comprehension process. ForUML will help CSE developers adopt better SE approaches for the development of their software. Similarly, software engineers who are not familiar with scientific principles may be able to understand a CSE software system just based on information in the generated UML class diagrams. 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 from Fortran code. 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 ForUML. We plan to add more diagrams to the website as we run ForUML on additional software packages. We believe that ForUML conforms to Chikofsky and Cross II [52] objectives of reverse engineering, which are identified as follows: (1) to identify the system's component and their relationships and (2) to represent the system in another form or at a higher level of abstraction.

In the future, we plan to address the limitations we have identified. We also plan to conduct human-based studies to evaluate the effectiveness and usability of ForUML by other members of the CSE 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 SE and CSE communities.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

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

References

[1] National Science Foundation, *Cyberinfrastructure for 21st Century Science and Engineering Advanced Computing Infrastructure (Vision and Strategies Plan)*, 2012, http://www.nsf.gov/pubs/2012/nsf12051/nsf12051.pdf.

[2] J. C. Carver, "Software engineering for computational science and engineering," *Computing in Science and Engineering*, vol. 14, no. 2, Article ID 6159198, pp. 8–11, 2011.

[3] J. H. Marburget, "Report of the high-end computing revitalization task force (hecrtf)," Tech. Rep., National Coordination Office for Information Technology Research and Development, 2004.

[4] V. K. Decyk, C. D. Norton, and H. J. Gardner, "Why fortran?" *Computing in Science and Engineering*, vol. 9, no. 4, Article ID 4263269, pp. 68–71, 2007.

[5] E. Akin, *Object-Oriented Programming via Fortran 90/95*, Cambridge University Press, Cambridge, UK, 2003.

[6] V. K. Decyk, C. D. Norton, and B. K. Szymanski, "Expressing object-oriented concepts in Fortran 90," *ACM SIGPLAN Fortran Forum*, vol. 16, no. 1, pp. 13–18, 1997.

[7] V. K. Decyk, C. D. Norton, and B. K. Szymanski, "How to support inheritance and run-time polymorphism in Fortran 90," *Computer Physics Communications*, vol. 115, no. 1, pp. 9–17, 1998.

[8] 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)*, vol. 7155 of *Lecture Notes in Computer Science*, pp. 367–376, Springer, Berlin, Germany, 2012.

[9] S. Filippone and A. Buttari, "Object-oriented techniques for sparse matrix computations in Fortran 2003," *ACM Transactions on Mathematical Software*, vol. 38, no. 4, article 23, 2012.

[10] K. Morris, D. W. I. Rouson, M. N. Lemaster, and S. Filippone, "Exploring capabilities within ForTrilinos by solving the 3D Burgers equation," *Scientific Programming*, vol. 20, no. 3, pp. 275–292, 2012.

[11] D. W. Rouson, J. Xia, and X. Xu, "Object construction and destruction design patterns in fortran 2003," *Procedia Computer Science*, vol. 1, no. 1, pp. 1495–1504, 2003.

[12] D. W. I. Rouson, H. Adalsteinsson, and J. Xia, "Design patterns for multiphysics modeling in Fortran 2003 and C++," *ACM Transactions on Mathematical Software*, vol. 37, no. 1, article 3, 2010.

[13] Z. Merali, "Computational science: ...Error," *Nature*, vol. 467, no. 7317, pp. 775–777, 2010.

[14] A. Nanthaamornphong, K. Morris, and S. Filippone, "Extracting uml class diagrams from object-oriented fortran: Foruml," in *Proceedings of the 1st International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering (SE-HPCCSE '13)*, pp. 9–16, Denver, Colo, USA, November 2013.

[15] J. C. Carver, "Report: the second international workshop on software engineering for CSE," *Computing in Science and Engineering*, vol. 11, no. 6, Article ID 5337640, pp. 14–19, 2009.

[16] G. V. Wilson, "What should computer scientists teach to physical scientists and engineers?" *IEEE Computational Science & Engineering*, vol. 3, no. 2, pp. 46–55, 1996.

[17] 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)*, pp. 550–559, Minneapolis, Minn, USA, May 2007.

[18] M.-A. Storey, "Theories, tools and research methods in program comprehension: past, present and future," *Software Quality Journal*, vol. 14, no. 3, pp. 187–208, 2006.

[19] M. J. Pacione, "Software visualisation for object-oriented program comprehension," in *Proceedings of the 26th International Conference on Software Engineering (ICSE '04)*, pp. 63–65, May 2004.

[20] J. Segal, "Professional end user developers and software development knowledge," Tech. Rep., Open University, England, UK, 2004.

[21] M. T. Sletholt, J. E. Hannay, D. Pfahl, and H. P. Langtangen, "What do we know about scientific software development's agile practices?" *Computing in Science and Engineering*, vol. 14, no. 2, Article ID 6081842, pp. 24–36, 2012.

[22] R. N. Britcher, "Re-engineering software: a case study," *IBM Systems Journal*, vol. 29, no. 4, pp. 551–567, 1990.

[23] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*, Addison Wesley Longman, Boston, Mass, USA, 1999.

[24] T. C. Lethbridge, S. Tichelaar, and E. Ploedereder, "The dagstuhl middle metamodel: a schema for reverse engineering," *Electronic Notes in Theoretical Computer Science*, vol. 94, pp. 7–18, 2004.

[25] OMG, OMG Model Driven Architecture (MDA), 1997, http://www.omg.org/mda/.

[26] Object Management Group (OMG), 1997, http://www.omg.org.

[27] 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)*, pp. 287–294, Denver, Colo, USA, April 2009.

[28] 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)*, pp. 187–191, Antwerp, Belgium, October 2008.

[29] 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 *Proceedings of the 13th Working Conference on Reverse Engineering (WCRE '06)*, pp. 297–298, Benevento, Italy, October 2006.

[30] E. Gansner, E. Koutsofios, S. North, and K.-P. Vo, "A technique for drawing directed graphs," *IEEE Transactions on Software Engineering*, vol. 19, no. 3, pp. 214–230, 1993.

[31] E. B. Duffy and B. A. Malloy, "A language and platform-independent approach for reverse engineering," in *Proceedings of the 3rd ACIS International Conference on Software Engineering Research, Management and Applications (SERA '05)*, pp. 415–422, Pleasant, Mich, USA, August 2005.

[32] V. K. Decyk, C. D. Norton, and B. K. Szymanski, "How to express C++ concepts in Fortran 90," *Scientific Programming*, vol. 6, no. 4, pp. 363–390, 1997.

[33] W. S. Brainerd, *Guide to Fortran 2003 Programming*, Springer, 1st edition, 2009.

[34] M. Metcalf, J. Reid, and M. Cohen, *Modern Fortran Explained*, Oxford University Press, New York, NY, USA, 4th edition, 2011.

[35] D. Rouson, J. Xia, and X. Xu, *Scientific Software Design: The Object-Oriented Way*, Cambridge University Press, New York, NY, USA, 1st edition, 2011.

[36] I. D. Chivers and J. Sleightholme, "Compiler support for the Fortran 2003 and 2008 Standards Revision 11," *ACM SIGPLAN Fortran Forum*, vol. 31, no. 3, pp. 17–28, 2012.

[37] J. Reid, "Coarrays in the next fortran standard," *SIGPLAN Fortran Forum*, vol. 29, no. 2, pp. 10–27, 2010.

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

[39] T. Mattson, B. Sanders, and B. Massingill, *Patterns for Parallel Programming*, Addison-Wesley Professional, 1st edition, 2004.

[40] J. L. Ortega-Arjona, *Patterns for Parallel Software Design*, John Wiley & Sons, 1st edition, 2010.

[41] M. Weidmann, "Design and performance improvement of a real-world, object-oriented C++ solver with STL," in *Scientific Computing in Object-Oriented Parallel Environments*, Y. Ishikawa, R. Oldehoeft, J. Reynders, and M. Tholburn, Eds., vol. 1343 of *Lecture Notes in Computer Science*, pp. 25–32, Springer, Berlin, Germany, 1997.

[42] A. Markus, "Design patterns and Fortran 90/95," *ACM SIGPLAN Fortran Forum*, vol. 25, no. 1, pp. 13–29, 2006.

[43] A. Markus, "Design patterns and Fortran 2003," *ACM SIGPLAN Fortran Forum*, vol. 27, no. 3, pp. 2–15, 2008.

[44] H. Neunzert, A. Klar, and J. Struckmeier, "Particle methods: theory and applications," Tech. Rep. 95-113, Fachbereich Mathematik, Universitat Kaiserslautern, Kaiserslautern, Germany, 1995.

[45] V. K. Decyk and H. J. Gardner, "Object-oriented design patterns in Fortran 90/95: mazev1, mazev2 and mazev3," *Computer Physics Communications*, vol. 178, no. 8, pp. 611–620, 2008.

[46] H. A. Muller, J. H. Jahnke, D. B. Smith, M.-A. Storey, S. R. Tilley, and K. Wong, "Reverse engineering: a roadmap," in *Proceedings of the Conference on The Future of Software Engineering*, pp. 47–60, Limerick, Ireland, June 2000.

[47] M. Fowler, *Refactoring: Improving the Design of Existing Code*, Addison-Wesley Longman, Boston, Mass, USA, 1999.

[48] T. J. Parr and R. W. Quong, "ANTLR: a predicated-LL(k) parser generator," *Software: Practice and Experience*, vol. 25, no. 7, pp. 789–810, 1995.

[49] 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)*, pp. 376–385, Florence, Italy, November 2001.

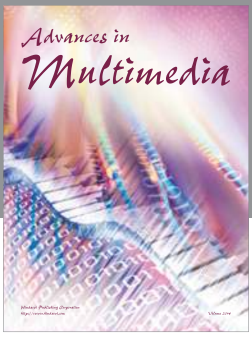
[50] A. Nanthaamornphong, K. Morris, D. W. I. Rouson, and H. A. Michelsen, "A case study: agile development in the community laser-induced incandescence modeling environment (CLiiME)," in *Proceedings of the 5th International Workshop on Software Engineering for Computational Science and Engineering*, pp. 9–18, San Francisco, Calif, USA, May 2013.

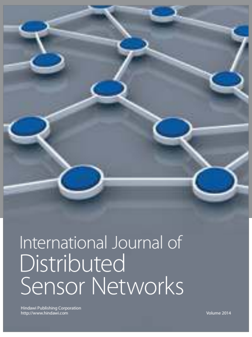
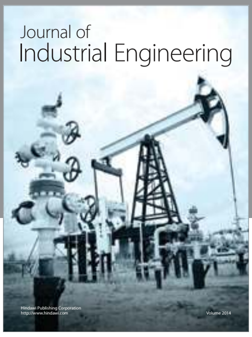
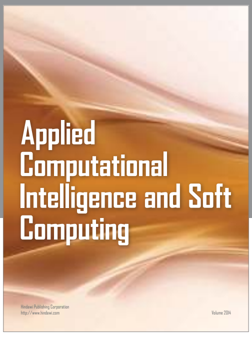
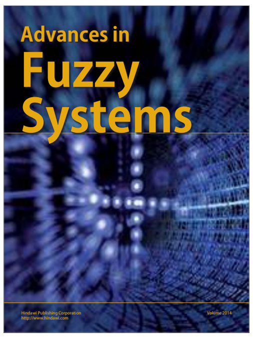
[51] L. C. Briand, Y. Labiche, and Y. Miao, "Towards the reverse engineering of UML sequence diagrams," in *Proceedings of the 10th Working Conference on Reverse Engineering*, pp. 57–66, Victoria, Canada, November 2003.

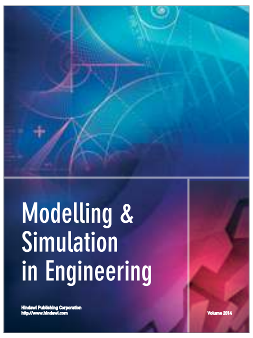
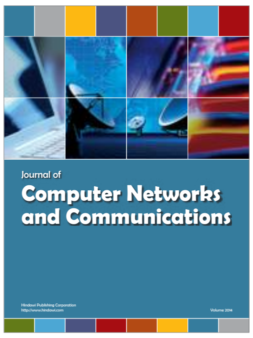
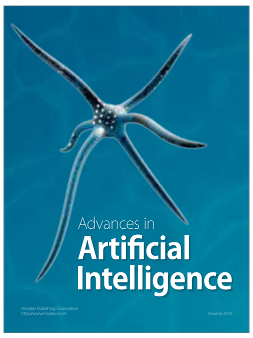
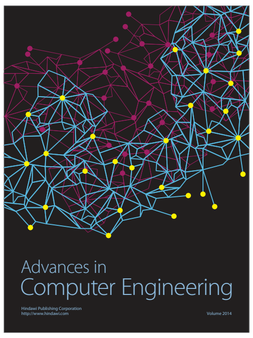
[52] E. J. Chikofsky and J. H. Cross II, "Reverse engineering and design recovery: a taxonomy," *IEEE Software*, vol. 7, no. 1, pp. 13–17, 1990.

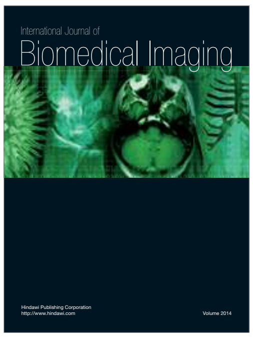
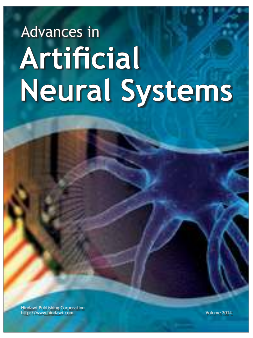
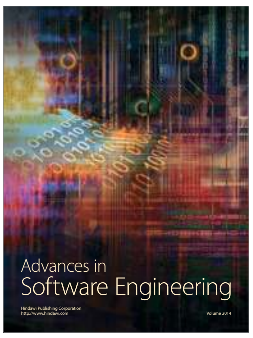
[53] IBM, *Rational Rose Enterprise*, 2013, http://www-03.ibm.com/software/products/en/enterprise/.

[54] Department of Computer Science University of Victoria, Rigi, 2001, http://www.rigi.cs.uvic.ca/rigi/blurb/rigi-blurb.html.

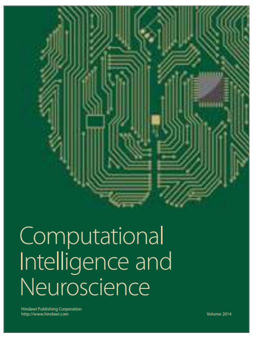

Advances in
Multimedia

The Scientific World Journal

International Journal of Distributed Sensor Networks

Journal of Industrial Engineering

Applied Computational Intelligence and Soft Computing

Advances in Fuzzy Systems


Modelling & Simulation in Engineering

Journal of Computer Networks and Communications

Advances in Artificial Intelligence

Advances in Computer Engineering

International Journal of Computer Games Technology

International Journal of Biomedical Imaging

Advances in Artificial Neural Systems

Advances in Software Engineering

Journal of Robotics

Advances in Human-Computer Interaction

Computational Intelligence and Neuroscience

International Journal of Reconfigurable Computing

Journal of Electrical and Computer Engineering