USING FORTRAN MODULES TO DESIGN AND DEVELOP MODULAR REACTOR ANALYSIS SOFTWARE COMPONENTS

M G. Mwaba
Atomic Energy of Canada Limited, Chalk River, Ontario, Canada

Abstract

This paper presents a design for heavy and light water property calculation routines using object-oriented design techniques. The designed routines are part of a new thermalhydraulics code being developed by Atomic Energy of Canada Limited (AECL). We demonstrate how application of object-oriented methodology leads to Fortran modules that use new features of Fortran 95 effectively. We also present performance metrics. This paper contributes in two ways. Firstly, it provides a methodology that can be used to systematically identify objects, assign responsibilities to the objects and establish the interaction among objects. Secondly, it shows how the designs can be communicated using the Unified Modeling Language (UML).

1. Introduction

The main objective of this paper is to present a methodology for designing Fortran modules using object-oriented techniques and suggest a way of communicating the design to other developers. A Fortran module is a programming unit introduced in Fortran 90. It can contain variables, parameters, derived type definitions, and procedures (subroutines and functions). Each of these entities can be declared as private or public. Private entities can only be used inside a module where they are defined. Entities declared with public attribute are available to any program unit that uses the module. Most developers view a Fortran module simply as a replacement for Fortran Common Blocks. As will be demonstrated in this paper, a Fortran module is a very powerful tool that can be used to design and develop modular software.

Most scientific software, including those for reactor analysis, has been coded in Fortran 77. Fortran 77 has been the language of choice for intensive numerical computational software due to its effective manipulation of arrays and its computational analysis capabilities. Fortran 77, however, has several limitations such as lack of data abstraction features, lack of dynamic memory allocation and lack of encapsulation. While some existing codes are well structured, the Fortran 77 limitations make it difficult to extend code (less suitable for developing modular software components) and lead to code that progressively becomes more difficult to maintain. These limitations have been addressed in the new standards (Fortran 90, Fortran 95 and Fortran 2003) making it possible for scientists and engineers to develop code that is re-usable, extensible and maintainable. Developing such code, however, requires a paradigm shift from functional decomposition techniques to object oriented techniques.

In the C++ community the object-oriented approach is one of the mainstream technologies used for the development of large-scale software systems. Trio_U [1], a thermalhydraulics software tool, is one example of a reactor analysis software designed and implemented using object-oriented techniques. In the Fortran community object-oriented technology is a new concept. Following the release of the modern Fortran standard, a measured interest in using object-oriented techniques for developing Fortran programs has been observed in literature [2-6]. Decyk et. al [2] have presented a

concise summary of the concepts of data abstraction, functional overloading, classes, inheritance, polymorphism, and how to implement them in Fortran 90. Machiel and Deville [3] have argued that complex data sets can be handled better with object-oriented programming and have shown how object-oriented concepts can be implemented in the framework of Fortran 90. Worth [4] has provided a survey of tools and techniques that allow Fortran programmers to use object-oriented programming techniques in the development of their programs. A good overview of how object-oriented features can be implemented in Fortran 90 is given by Akin [5]. Fortran 95 has added more desirable features and examples of how they can be used can be found in [6]. While these papers do an excellent job at showing how the new concepts can be implemented in Fortran, information on how to design the routines and how to communicate such designs is very scarce. Worth mentioning is the paper by Gary and Roberts [7] that discusses both the design and implementation of Fortran 90 object-oriented features. Missing from this paper is how best to communicate the designs.

The greatest challenge facing a Fortran developer is to come to speed on how to create object-oriented designs and how to communicate such designs. This paper contributes in two ways. Firstly, it provides a methodology that can be used to systematically identify objects, assign responsibilities to the objects and establish the interaction among objects. Secondly, it shows how the designs can be communicated among developers and other stakeholders using the Unified Modeling Language (UML) [8]. UML is a general purpose visual modelling language used to specify, visualize, construct and document major elements of a software system. While UML is very popular among C++ developers, the author is not aware of any published work in the public domain on the use of UML by Fortran developers. This paper attempts to fill that void.

This paper is organized as follows. A brief background of the problem is outlined in Section 2. The proposed design methodology is explained in Section 3. To consolidate the ideas of Section 3, design of routines for heavy and light water property evaluations for use in a new advanced thermalhydraulics network analysis code is presented in Section 4. Section 5 discusses the implementation and presents the performance metrics. The concluding remarks are given in Section 6.

2. Background

The CATHENA 4 code is the next generation systems thermalhydraulics analysis code currently being developed by Atomic Energy of Canada Limited (AECL) for simulating transient, non-equilibrium, multi-phase flow and heat transfer. The CATHENA 4 code will primarily be used in the design and safety analysis of CANDU reactors and thermalhydraulics experimental facilities. Object Oriented Design (OOD) techniques have been used for the design of CATHENA 4 so as to promote a modular design thus providing ease of future extensions to support multi-dimensional applications and the consolidation of other AECL analysis codes. The CATHENA 4 code is designed to comprise several subsystems, each subsystem being an aggregate of modules. Each module has been designed to contain data elements bundled together with procedures that operate on the data. One of the modules, known as HLWP 2.0, is responsible for the calculation of properties for heavy and light water and some non-condensable gases at given conditions. The design of HLWP 2.0 is described in this paper.

Thermodynamic and transport properties of fluids can be estimated by using a variety of fitted functions. To maintain mathematical consistency between the thermodynamic properties and their derivatives, the single-state dependent property of specific entropy S as a function of absolute

pressure P and specific enthalpy h is an appropriate choice for fitting via piecewise Hermite polynomials [9]. All the required properties can then be obtained from the first and second derivatives of specific entropy. These thermodynamic property expressions can be derived from the fundamental relations for a simple, compressible system of fixed chemical substances [10].

3. Design Methodology

The design philosophy adopted is to utilize the Fortran module as the fundamental programming unit. Modules are designed to take advantage of the Fortran 95/2003 features, such as allocatable arrays, pointers, private and public attributes, optional arguments and derived types [11]. With these features desired design goals such as readability, portability, extensibility and maintainability (ability to fix errors in the code) can easily be demonstrated. A module can contain variables, parameters, derived type definitions, and procedures (subroutines and functions). Each of these entities can be declared with a private or public accessibility attribute. Private entities can only be used inside a module where they are defined. Entities declared with public attribute are available to any program unit that uses the module. An important aspect of a module is its ability to facilitate data encapsulation. The module is designed to contain one or more derived data types and the procedures that operate on these data types.

The modules are designed using an object-oriented approach. The general steps taken to design a module using an object-oriented approach are outlined below:

1. Objects that can contribute to performing the task of the subsystem are identified. Each such object is assigned a responsibility, or a set of related responsibilities.
2. Information needed for the object to perform the assigned responsibility is identified. This information constitutes the data to be managed by the object.
3. The behaviour of the object is next assigned. The behaviour is a set of tasks that the object should perform in order to fulfill its responsibilities.
4. Associations are established to indicate how objects relate to one another.
5. Decisions are made on what information an object is "willing" to share with other objects. The information to be shared is classified as public while that not to be shared is denoted as private.
6. The means by which an object will communicate with others is defined. This forms an interface of the object.

Performing steps 1 – 6 naturally leads to a Fortran module in the following manner:

- The information required for the object to perform its tasks becomes the data context of the module (Step 2).
- The behaviour of the object translates into procedures (subroutines and functions) within the modules (Step 3).
- Associations between objects give information on how they interact with one another (Step 4).
- Decisions on information sharing between objects lead to assignment of private and public attributes within the modules (Step 5).
- The means of object communication defines a module's interface (Step 6).

UML standard defines several diagrams for use in communicating software elements. For this paper, the design can then be presented using two types of UML diagrams, namely, class diagrams and sequence diagrams. A class diagram is a static view representation of the elements that make up the software. A sequence diagram is a dynamic view representing how the software elements interact. Modules and their relationships are presented in terms of class diagrams. A complete class diagram includes attributes, procedures (subroutines and functions) and associations. The interaction of modules is shown using a sequence diagram.

4. HLWP 2.0

The problem to be solved can be stated as follows: We need a set of routines for evaluating thermodynamic and transport properties of light water (H_2O) and heavy water (D_2O), and thermodynamic properties of non-condensable gases. Hermite polynomial fitting methods are used for the calculation of H_2O and D_2O properties using entropy and its derivatives as a basis for the fits to available standard property data. The data for non-condensables is fitted using standard generating functions. The property data is to be loaded at run time.

4.1 Requirements

To have efficient functioning software, several requirements have to be satisfied. Here we highlight just four of them.

1. Speed is a key requirement since for two-phase flow simulation fluid properties are evaluated at each time step.
2. Allowance should be made for future expansion of the property database, i.e adding more materials or expanding the property data set.
3. Property routines can be used in a variety of thermalhydraulics codes, e.g. system codes and subchannel codes. Therefore abstraction and re-usability are key requirements.
4. The HLWP 2.0 should be able to run in parallel when required.

4.2 Design

Our efforts have been directed at specifying Fortran modules and encapsulating data. By following this method, we ensure that Fortarn modules ‘own’ data leading to data localization. Following the design methodology presented in Section 3 several core and support modules are identified. In this paper we show and describe five core modules and three support modules. The five core modules are *FldProps_m*, *Entropies_m, Hermites_m, PropsData_m* and *Reader_m*. The three modules playing support roles are *PosLOcate_m, KindType_m* and *Iounit_m.* The relationship between these modules is depicted in a class diagram shown in Fig. 1 in conformity with the UML notation. In the figure, each rectangular box represents a Fortran module. The three parts of the box show the module’s name, its data attributes (variables and parameters) and the procedures that act on the data. Private attributes are preceded by a negative sign while public ones are preceded by a positive sign. The open diamond indicates the Fortran “use” relationship. It allows a Fortran module to have access of the public attributes of another module. For instance, since *FldProps_m* uses *PropsData_m*, *FldProps_m* has access to the variable PropData_t and to the subroutine Get_data().

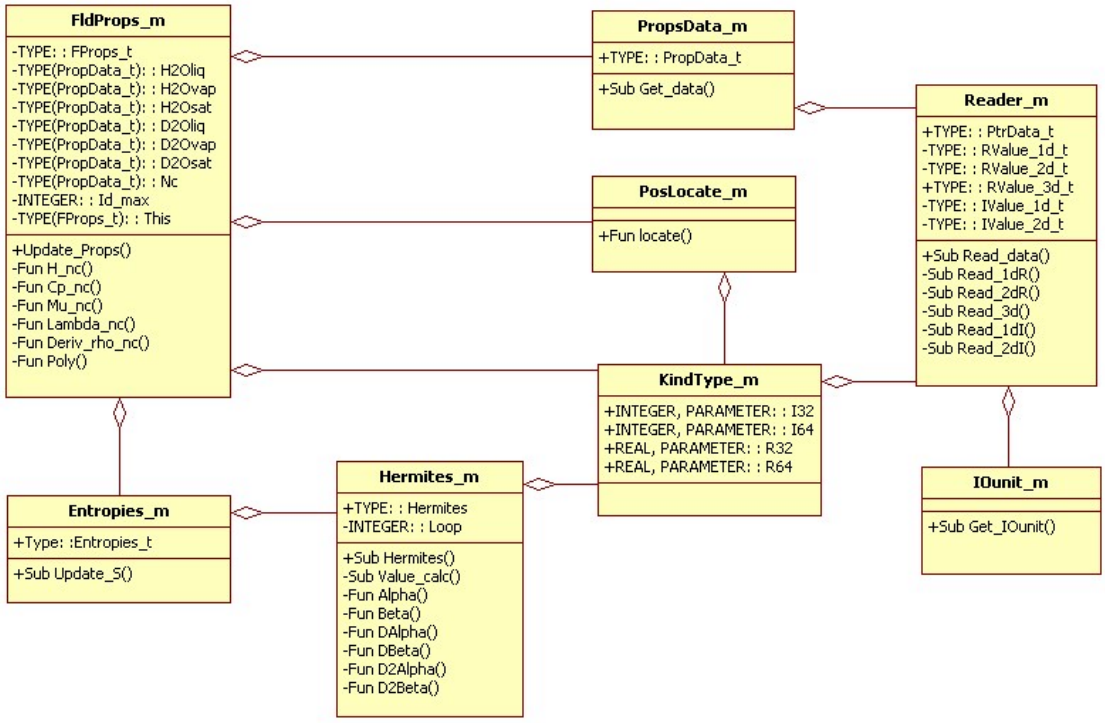

Figure 1 Class diagram showing the relationship between modules making up HLWP 2.0.

FldProps_m is the entry point to HLWP 2.0 and is the only module that interacts directly with a client. The main purpose of *FldProps_m* module is to calculate thermodynamic and transport properties of fluids given pressure and temperature (or enthalpy). To satisfy the requirement for speed, minimization of communication among modules and within modules is a key design goal for this module.

FldProps_m contains one derived data type and nine procedures. The derived data type has a public attribute so that client programs can access the properties. Seven of the procedures have private attribute and only one procedure has a public attribute. The public procedure provides the module's interface allowing external programs to access it. Some properties are calculated using entropies and their slopes. *FldProps_m* relies on *Entropies_m* to provide this information. *Entropies_m* has one derived data type and one procedure. Entropies and their slopes are calculated using hermite values. The responsibility of calculating hermite values is assigned to *Hermites_m*.

The module *PropsData_m* is responsible for retrieving data required for property calculations from storage sources. *PropsData_m* contains one derived data type and one procedure. The data type has public access attribute while the procedure has a private attribute. The responsibility of reading data is assigned to *Reader_m*. This module contains subroutines that read data from files and store it in an array of pointers. The files contain hermit coefficients that are used in the calculation of fluid properties.

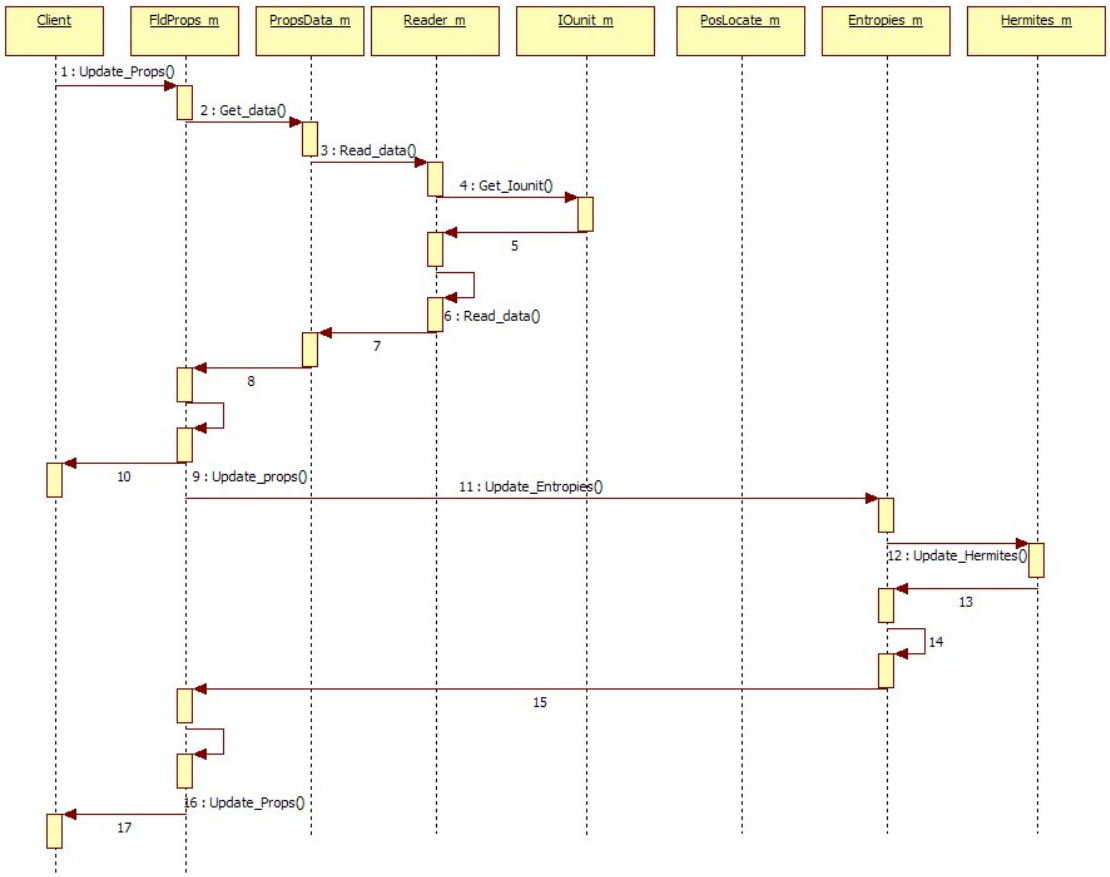

Figure 2 Sequence diagram depicting the dynamic interaction of objects in HLWP 2.0.

The dynamic interaction of modules within HLWP 2.0 is shown in Fig.2. A client (program that needs fluid properties) sends a message "Update_Props" to *FldProps_m*. Sending a message is analogous to making a call to a procedure. If property data is not yet loaded, *FldProps_m* sends a message to *PropsData_m* requesting for data. If data is available, *FldProps_m* requests for updated entropy values from *Entropies_m.* To update the entropy values *Entropies_m* asks *Hermites_m* to update the hermites values. Once the hermite values are updated, *Entropies_m* calculates the new entropies, the derivatives and informs *FldProps_m*. Once *FldProps_m* has the required information, it calculates the properties and sends the updated properties to the client.

The design presented above is modular and extensible, just to mention two attributes. It is modular since each of the modules is independent and interacts only through the interface. The advantage of this is that separate modules can be assigned to different developers. The modular nature also allows for extensibility. If in future a new material is needed, all we need is a new property data file. If changes were needed they would be minimal and localized.

5. Implementation and Performance

The design outlined above has been implemented in Fortran 95 and compiled using the Intel 10.1 compiler. Figure 3 shows an excerpt of the *Hermites_m* implementation. Only the subroutine *Hermites* and the derived type *Hermites_t* and are accessible from outside the module. Of interest is the fact that the components of *Hermites_t* are private and therefore not accessible from outside the module. With this implementation we can add or subtract components to *Hermites_t* without affecting the sub-programs that use *Hermites_m*.

```
MODULE Hermites_m
!
!
!
  USE KindType_m     ! Module being used by Hermites_m

  PRIVATE   :: Values_calc   ! Subroutine accessible only within Hermites_m
  PRIVATE   :: Alpha   ! Function accessible only within Hermites_m
  !
  !
  !
  PUBLIC   :: Hermites   ! Subroutine accessible outside of Hermites_m

  TYPE, PUBLIC :: Hermites_t   ! Derived data type.  Can be declared outside of Hermites_m
     PRIVATE
      REAL(KIND=R64) :: A  ! Component of Hermite_values_t. Accessible only by procedures within
                                    !Hermites_m
      REAL(KIND=R64) :: B    ! Component of Hermite_values_t. Accessible only by procedures within
                                    !Hermites_m
      INTEGER(KIND=R64) :: C    ! Component of Hermite_values_t. Accessible only by procedures within
                                      !Hermites_m
 END TYPE Hermites_t


CONTAINS

  SUBROUTINE Hermites (……..)

   !
   !

  END SUBROUTINE Hermites

  SUBROUTINE Values_calc (…..)

   !
   !

  END SUBROUTINE Values_calc

  FUNCTION Alpha (…..)  RESULT (…)

   !
   !

  END FUNCTION Alpha

END MODULE Hermites_m
```

Figure 3 Excerpt of *Hermites_m* implementation

Most Fortran developers are concerned about the performance codes implemented using object-oriented techniques. A run-time speed comparison was made between HLWP 2.0 and HLWP 1.0, a Fortran 77 implementation. The tests were run on a Pentium 4 2.80 GHz 504 MB RAM machine. The results are shown in Table 1.

Table 1: CPU times for HLWP 1.0 and HLWP 2.0 comparison

Nodes	CPU Time (sec)	
	HLWP 1.0	HLWP 2.0
1000	15.20	16.10
10000	166	178
100000	1884	2062

It can be seen that on average the speed of the new implementation is 0.90 times that of the Fortran 77 implementation. The slight reduction in speed is due to the overheads associated with OOD. The performance is acceptable considering the many benefits associated with modular software.

6. Concluding Remarks

Fortran now has features that support the development of elegant, modular software. A Fortran module is one such feature. We have presented a methodology that can be used to develop modular software using OOD. We have also shown that Fortran designs can be communicated using UML. The method has been applied to the design of heavy and light water property calculations routines for use in new reactor analysis software. We conclude that the method presented in this paper can assist the Fortran scientific community develop well-designed modules whose implementation produces code that exhibits desired features such as maintainability, re-usability and extensibility.

7. References

[1] C. Calvin, O. Cueto and P. Emonot, “An Object-oriented approach to the design of fluid mechanics software”, Mathematical Modelling and Numerical Analysis, Vol. 36, Iss. 5, 2002, pp. 907-921.

[2] V.K. Decyk, C.D. Norton and B.K. Szymanski “Introduction to Object Oriented Concepts using Fortran 90”, Technical Report UCLA IPFR Report PPG-1560, 1996.

[3] L. Machiels and M.O. Develle, “An Entry to Object-Oriented Programming For the Solution of Partial Differential Equations,” ACM Transactions on Mathematical Software, Vol. 23, Iss. 1, 1997, pp. 32-49.

[4] D. J. Worth, “State of the Art in Object Oriented Programming with Fortran”, Science and Technology Facilities Council, Technical Report, RAL-TR-2008-002, ISSN 1358-6254, 2008.

[5] J.E. Akin, “Object Oriented Programming via Fortran 90,” Engineering Computations, Vol. 16, Iss 1, 1999, pp. 26-48.

[6] J. Mahseredjian, B. Bressac, A. Xémard, P.-J. Lagacé and P. Lacasse, “A Fortran-95 Implementation of EMTP Algorithms”, Proceedings of the International Conference on Power Systems Transients 686–691, Rio de Janeiro, Brazil, 2001, June 24-28.

[7] M.G. Gray and R.M. Roberts, “Object-based Programming in Fortran 90,” Computers in Physics, Vol. 11. Iss. 4, 1997, pp.355-361.

[8] T. A. Pender, UML Weekend Crash Course, Wiley Publishing Inc., New York, 2002.

[9] Y. Liner, B.N. Hanna and D.J. Richards "Piecewise Hermite Polynomial Approximation of Liquid-Vapour Thermodynamic Properties", Fundamentals of Gas-Liquid Flows, Vol. 72, 1988, pp. 99-102.

[10] K. Wark, Thermodynamics, Third Edition, McGraw-Hill Book Company, New York, 1977.

[11] S. Chapman, Fortran 95/2003 for Scientists and Engineers, McGraw-Hill, New York, 3rd Edition, 2007.