REENGINEERING AND REFACTORING LARGE-SCALE SCIENTIFIC PROGRAMS WITH THE UNIFIED PROCESS: A CASE STUDY WITH OSIRIS PIC PROGRAM

Jincheol B. Kim[1,2], In Soo Ko[1] , Hyyong Suk[2]
1. POSTECH, Pohang, Kyoungbuk, 790-784, South Korea
2. Center for Advance Accelerators, Changwon, Kyoungnam, 641-120, South Korea

Abstract

OSIRIS is a large-scale particle-in-cell (PIC) code which was developed at Particle Beam Physics Laboratory (PBPL) in UCLA for researches of laser-plasma interactions. OSIRIS was reengineered and documented in UML by our group and ported to Linux cluster machine of 8 nodes. We report our current status of developing the extended version of OSIRIS, which was named as OSIRIS-X and maintained and developed with the Unified Process. Some guidelines in designing and refactoring large-scale scientific codes are presented and discussed. A design model of numerically intensive programs for large-scale computing is suggested, and it is discussed how we can use it for rapid development and prototyping of scientific programs. We also discuss future challenges and prospects in OSIRIS-X development.

INTRODUCTION

Computational tools and codes in accelerator researches become more important. They tend to become larger and more complex as accelerator technologies advance. Many of the tools have become commercialized these days and most of the problems with current accelerator technologies can be solved by these commercialized and generalized softwares in general. But when new concepts and its principles must be explored, a wholly new approach and problem-solving and design technique must be sometimes introduced. This initiation of a new challenge in accelerator researches requires development of appropriate computation and simulation codes to investigate them in an appropriate way.

Though software engineering has advanced much as the demand rises on cost-efficient methodologies in maintenance and development of complex softwares, many of the computation and simulation codes are still based on traditional programming techniques. Even though some of the recent advanced concepts such as Object-Oriented Programming (OOP) have been introduced into the code developments, powers of those advanced concepts are not facilitated fully.

OSIRIS is a particle-in-cell simulation code for researches of laser-plasma interactions and advanced accelerator concepts developed by Particle Beam Physics Laboratory (PBPL) in UCLA. It was imported to Center for Advance Accelerators in Korea Electrotechnology Research Institute for laser-plasma researches and has been used to analyze and investigate the experiments in the group theoretically. It is currently under upgrade and reengineering with the name of OSIRIS-X (OSIRIS eXtended) to include new physical features into the code for future researches.

We report our progresses on the introduction of concepts of object-oriented software engineering and design patterns into scientific code developments. A design model based on science activities in real world is suggested based on our experiences of refactoring OSIRIS. Some practical issues on effective implementations are discussed.

DESIGN MODEL AND PRINCIPLES

Software Model based on Real-World Situations

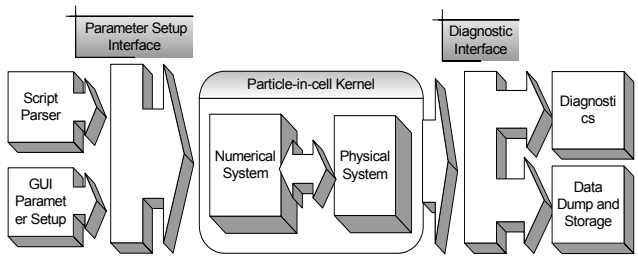

Figure. 1. Design Model of OSIRIS-X

OSIRIS-X has the design model based on science activities in real world, which is described in Fig. 1. PIC Kernel, which is the original OSIRIS PIC code, simulates the complex interactions between electromagnetic field and charged particles as a time evolution of the machine of continuous state variables such as x, p, E, B, ρ, and J. Initial parameter setup is delivered to the kernel through a common interface. With this common interface to the kernel, different types of input parameter specification can be implemented, appended, and modified without affecting the kernel.

Diagnostic components also communicate with the kernel through a common diagnostic interface. This interface specifies a standard communication channel between the diagnostic module and the PIC kernel. This diagnostic interface only provides the diagnostic module with basic information on the status of the physical variables. Calculations of physical quantities and analysis of the simulation are implemented in the independent diagnostic module and the data dump and storage module. This imitates the measurement in real world.

Based on the design model in Fig. 1, class hierarchies and relations were rearranged and restructured as shown in Fig. 2. Each component in Fig. 1, such as PIC kernel and Diagnostics, is designed as a subsystem with classes

supporting its function. In the previous version of OSIRIS, the classes were not organized as subsystems in a consistent design model and complex interdependencies between classes and subcomponents made the modification of OSIRIS difficult. We introduced interface classes between subsystems to maintain low coupling between them. It makes the modification of parts of the program much easier without affecting other components. Subsystems communicate with other subsystems through the standardized interface implemented as interface classes. Any modified features can be easily added if a new module follows the defined interfaces as the communication channel to the subsystem. These interface classes use patterns, which is the standardized structures and idioms of object-oriented software design, to maintain low coupling between subsystems and high cohesion among the classes inside each subsystem. Some basic patterns for these interface classes will be discussed later.

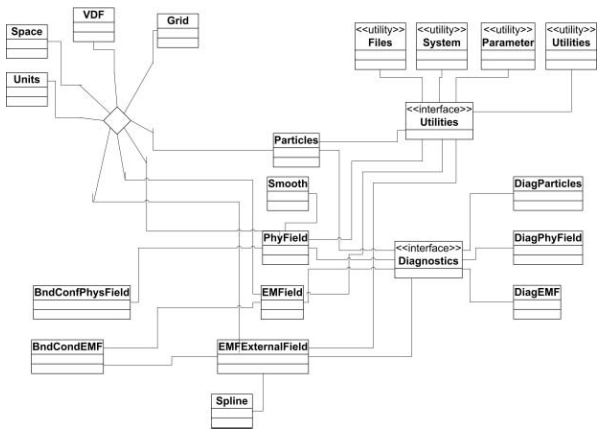

Figure 2. Class Interdependency and Organization with Interface Classes

Performance Issues and Its Relation to Design Model

The object-oriented design and analysis (OODA) is generally suitable for rapid and systematic development of large scale softwares, but performance can be seriously deteriorated because of layered structures and heavy communications between classes and components. Scientific softwares require high performance and this constraint must be considered when we design softwares of heavy computations such as PIC simulation codes. Some design principles are suggested as follows.

- Low coupling must be maintained between subsystems which are not related to computations, such as script interpreter and diagnostics and data dump modules. This makes software maintenance, enhancement and addition of new functions much easier without affecting other components. This low coupling can be easily introduced to the model by using design patterns [2, 7].
- Layers and hierarchies must be reduced as many as possible in modules of heavy computations. We must keep simple structures in components related to heavy computations to reduce process loads due to communications between structures.
- When we design the numerical kernel, it is better to design several separate kernels which deals with specific cases efficiently and to make an appropriate kernel loaded suitable for a problem of specific conditions and constraints. One large kernel which calculates all of the general cases usually deteriorates the performance. For example, it is more beneficial in performance to design kernels with Cartesian coordinate and spherical coordinate separately and to load each kernel for specific problem geometry than to design a kernel which solves a problem with generalized coordinates and meshes. Using small components suitable for specific problems is the general strategy adopted in the design of operating systems like UNIX, such as the Mach microkernel for example, and performance critical systems.

Use of Design Patterns

Design pattern is descriptions of structures and communicating objects and classes that are customized to solve a general design problem in a particular context which occurs repeatedly in the software design [7]. We can develop softwares consistently and organize and view the whole structure of complex softwares by using design patterns. Maintenance and iterative enhancement of large scale scientific programs can be made easier and more systematic by using design patterns.

The interface classes which connect different subsystems with PIC kernel were designed by using basic design patterns. It makes OSIRIS-X easily extensible. Some basic patterns are suggested as follows:

- A small kernel for specific problems is better for performance enhancements. The program must be able to select and load an appropriate kernel to start a new simulation with maintaining low coupling with other subsystems. Adaptor, Factory, Strategy and Singleton design patterns make this possible.
- One of the objectives introducing interface classes is to make them more efficient to extend existing modules and to append a new module simply by using standard methods defined in interface classes. Some modules such as diagnostic modules can include much different functions and implementation strategies and this can make it difficult to use the standard methods. Facade and Composite patterns are appropriate for this design problem.

ISSUES ON IMPLEMENTATION

Use of CASE Tools

Appropriate CASE tools can greatly enhance the productivity of software design and development. Rational Rose™ is a well known and widely used OODA case tool but it is expensive. CASE tools such as Rational

Rose™ is essential for developing large-scale and enterprise-level applications with appropriate team managements. It is sometimes enough for scientific software design such as OSIRIS-X to use public sharewares, such as Objecteering UML Personal Edition, or GNU public softwares such as Umbrello, which is laboratory-level and 3 to 5 people are engaged in maintenance and development.

Objecteering UML Personal Edition is provided freely in the Internet, and Professional Edition and Enterprise Edition are also provided with relatively low prices than other commercial CASE tools like Rational Rose™. Objecteering UML Personal Edition does not support team management and some features of the Unified Process, but it is useful enough for our purposes. Umbrello, a GNU public-licensed CASE tool, is small and practical. Umbrello also supports source code generation from the design model by UML, which is very useful and can make the development process much efficient by providing source codes containing class definitions which implements complex class interdependencies consistently. The Microsoft Visio 2002 supports UML modelling and some project management features which is useful for documentation and design of OSIRIS-X.

Embedding Python Interpreter in OSIRIS-X

Scientific softwares usually set up initial parameters of a simulation or a computation by reading a simple script file. Programs written in FORTRAN 90 or 95 use the namelists for parameter setup. Programs written in C or C++, such as XOOPIC, include their own lexical analyzer and parser by using powerful tools such as LEX and Yacc or Bison. The namelist feature in FORTRAN 9x is suitable for specifying initial parameters but it is restricted and difficult to set up parameters dynamically. An independent script interpreter can be an alternative but it takes some time to develop even though many tools helping programming language design are now available. A solution to this problem is to embed an existing script-language interpreter into the program and to let the embedded interpreter interpret the input script and transfer the interpreted values to the numerical kernel.

Python is a concise and powerful object-oriented script language and has widely used in various applications. Python can be embedded in application programs or any application can be imported as a module of the Python environment by API's provided by Python. Many of the powerful features of Python, such as dynamic typing, powerful data types such as list and tuples, and various classes and functions of Python libraries, can be used in writing the parameter script and in user own applications by embedding the interpreter in an user's application.

The previous version of OSIRIS used the namelist script of FORTRAN 9x language. Python interpreter will be embedded in OSIRIS-X which will make researches by OSIRIS with many complex parameters much efficient. Tkinter, Python's GUI module, will be integrated to OSIRIS-X to help users set up parameters, to get diagnostic information on the simulation and to post-process large amounts of data produced by the OSIRIS PIC kernel. The PIC kernel of OSIRIS-X will also be developed as a Python module and then interactive simulations will be possible which will be useful in initial parameter search for finding meaningful parameter range.

FUTURE PROSPECTS

The suggested design model and principles can be adopted generally in developments of various codes for accelerator physics researches. Based on the design model and principles, OSIRIS-X will be upgraded for better computational performance and for conveniences which helps efficient data production and analysis with GUI-based parameter setup and diagnostics. Additional PIC kernels of OSIRIS-X are under development independently from PBPL of UCLA. It will contain PIC kernels with supports of generalized coordinates of grids, particle motions and field profiles, standard diagnostic interfaces for easy extension of diagnostic modules, new physics modelling modules for the kernel. Scripting input parameters with Python will help users exploit the powers of Python language in their simulations and enhance the productivity of simulation researches.

ACKNOWLEDGEMENTS

This work was supported by the Korea Research Foundation Grant (KRF-2003-015-C00121). J.B. Kim and I.S. Ko also appreciate the financial support from the Center for High Energy Physics at Kyoungpook National University. H. Suk was supported by the Creative Research Initiatives Program of Korea Ministry of Science and Technology.

REFERENCES

[1] Roy Gerrit Hemker, Particle-in-cell Modelling of Plasma-Based Accelerators in Two and Three Dimensions, Ph. D. Dissertation, University of California, Los Angeles, 2000.

[2] Craig Larman, Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process (2nd Edition), Prentice-Hall PTR, 2002.

[3] J.P. Verboncoeur, A.B. Langdon, and N.T. Gladd, Comp. Phys. Comm. 87, 199-211, 1995.

[4] C.D. Norton, B.K. Szymanski, and V.K. Decyk, Communication of the ACM, vol. 38, No. 10, Oct. 1995, pp. 88-100.

[5] Grady Booch, Object-Oriented Analysis and Design (2nd Edition), Addison Wesley Longman, Inc., 1994.

[6] Grady Booch, James Rumbaugh, and Ivar Jacobson, The Unified Modeling Language: User Guide, Addison Wesley, 1999.

[7] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley, 1995.