Object-Oriented Programming Hardware/Software Supports and Comparison

Junyi Li
Department of Computer Engineering
Dongguan Polytechnic
Dongguan, China
E-mail: lijunyi68@126.com

Yuhua Zhang, Zhenkun Li
Faculty of Computer
Guangdong University of Technology
Guangzhou, China
E-mail: gdutzyh@163.com

Anthony S. Fong
Department of Electronic Engineering
City University of Hong Kong
Kowloon Tong, Hong Kong
E-mail: anthony.fong@cityu.edu.hk

***Abstract*—Object-oriented programming has advantages of reusability, more extensibility, and easier maintainability. While it needs hardware/software environment to support the OO behaviors. In the paper, three OO supporting mechanisms are discussed, the advantages and disadvantages are compared.**

Keywords - object-oriented programming; compilation; virtural memory; virtual machine; object machine

I. INTRODUCTION

Object-oriented programming produces greater reusability, more extensibility, and easier maintainability [1]. Modern applications are developed with object-oriented programming languages. In traditional systems, processor architecture like Complex Instruction Set Computer (CISC) or Reduced Instruction Set Computer (RISC) [2] does not support object manipulation. To support object technology in nowadays system, different methods are proposed. Methods can be roughly categorized into three kinds. They are: compiling objects into native, developing of software based object virtual machine, and developing of hardware based object machine.

In this paper, we will go through these methods using in current systems for the support of object-oriented programming. The pros and cons of each method are discussed.

II. OBJECT-ORIENTED PROGRAMMING AND ENCAPSULATION

A. Basic Components in Object-oriented Languages

There are three basic components in the object-oriented languages, which are classes and instances, instance variables/fields and methods.

A Class defines properties and operations that are common to a collection of objects. It can also be viewed as a program structure or module that it contains data/variables and operations. Classes are templates for creating objects and cannot manipulate at runtime environment. Therefore, an "object" is an instance of classes of runtime entities in the object-oriented software. Since classes only define what are common to all of their instances. Every instance (object) has it own identity and a lifetime within the execution of the program.

Data/variables in a class are known as instance variables or fields. Individual attributes of an object can be described by instance variables within that particular object. The values of those instance variables can be viewed as the state of an object. When updating the values of these instance variables, the state of a particular object is changed. Instance variables of and object can be visible or invisible from other objects. The details of the visibility will be described in the following section.

Another component of an object is method. Methods are procedures or functions used to describe the behaviors associated with an object. They represent the actions that can be performed by an object or on an object. Since executing a method can change the value of the attributes or the object, the state of the object can also be changed. Therefore it can also act as a state transformer. Each method has a name and body that performs the action associated with the method name. When a caller object invokes a method of another object, that particular method will be performed and the result will be passed back to the caller object. This is a process of method invocation.

B. Slot Model and Encapsulation

In [3], the definition of class contains the definition of slots. There are two types of slots: one is used to hold the data and the other is used to hold the procedural code that implement the operations defined for the class. When the object creates, the space of each slot is allocated by the definition of the class. Each object has it own slots to hold the data and operations. The class definition of slot model is shown in Figure 1.

Published by Atlantis Press, Paris, France.

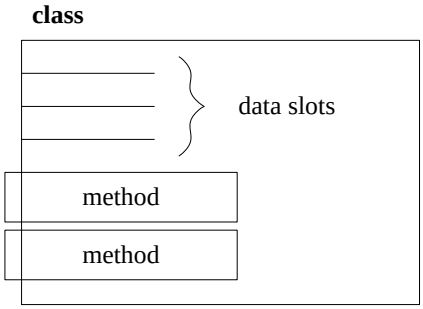

Figure 1. Class definition of slots model

Data slot can be a constant or defined as instance variable that can be changed by the program in runtime. Therefore, data slot can be read-only or read-write. When the data slot is intended to be read-only, only read operation can be performed to that data slot. Then, it is a constant in the object. When the data slot intended to be read-write, it can be modified and the state of the object is changed.

Method slot contains methods are usually considered to be read-only [4]. The meaning of read-only is just called a method or to pass it as a parameter. Therefore, method slot can be considered to be execute-only.

When the program wants to access a data slot in the direct fashion, the statement can be wrote:

i.s

i is referred to an instance of a class and *s* is referred to a particular slot. If the program wants to modify a data slot s for instance *i* by a value *newvalue*, the statement can be wrote:

i.s = newvalue

A direct modification can be performed with the above statement. This approach is a simple method to access slot in an instance, but the internal structure of the instance is directly accessed or modified. A class should remain encapsulated and hide the internal structure and implementation from outside. The process of hiding the internal details of a class is called encapsulation.

Encapsulation ensures object cannot affect the internal implementation of another. Once a class is defined, the data slot of the class should be accessed though its external interfaces only. The external interface of a class is represented by a group of operations and it defines the visible parts for other objects and which operations can be performed of that class. The internal details of a class may consist of data slots and methods. The internal data slots and methods are hidden from outside. Other objects are unable to understand what the class represents and how it operates. An encapsulated class is shown in Figure 2.

For example, when building a stack type, the external interface is the operation of push, pop and test the stack is empty or not. In the class of the stack type, we can use different internal data structure to hold the stack elements such as vector or link list. Elements and the implementation of the stack are hidden from the external view. Other objects can only access the stack though the interfaces that are methods for push and pop. If the implementation of the stack type is changed, programmer is required to rewrite the external interfaces for the new implementation. Since the external interface is the same, changing internal implementation will not affect the external caller. It can reduce the interdependencies between objects, so that the reliability of the software system is improved.

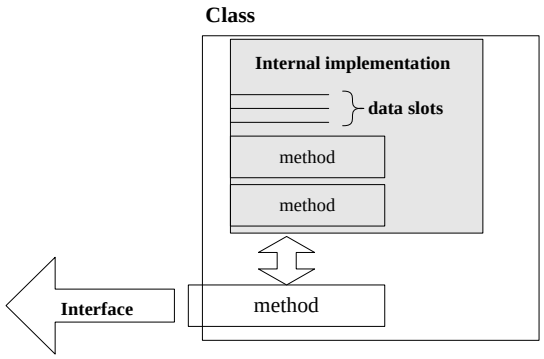

Figure 2. An encapsulated class

III. VON NEUMANN HARDWARE VS. OBJECT-ORIENTED HARDWARE

Von Neumann hardware [2] is designed for procedural programming while Object-Oriented hardware [4,5,6] is designed for object-oriented programming. The concept of procedural programming is based on module and scope. Module is a collection of procedures. On the other hand, the concept of object-oriented programming is based on self-contained objects. Objects are key elements to model the system.

In traditional von Neumann hardware, data can be accessed by its address within the same addressing space. There is no protection for individual data except what is presented through memory management. The deficiency of data protection does not affect the performance of procedural programming, but removes the security nature of object-oriented programming at architectural level. Data security support is therefore implemented in hardware. There are various ways to implement protect through logic, such as hardware access privilege tables to control access. Descriptor could be a way to store access modifiers for individual data. Object-oriented hardware is able to use information in descriptor to provide access checking.

For control flow, the direct branching to subroutine in procedural programming is different with the indirect branching to methods in object-oriented programming. The indirection in object-oriented programming may lead to a redesign of pipeline stages. For intra-method branches, boundary checking is performed to prohibit codes to access outside method space. Automation of stack management is used in object-oriented hardware in order to minimize the chance of programming bugs due to mistaken programming.

IV. COMPILATION APPROACH

In the compilation approach, the system consists of a traditional processor, a traditional operating system, and some application process running in different address spaces. Figure 3 shows the architecture of a traditional computing system.

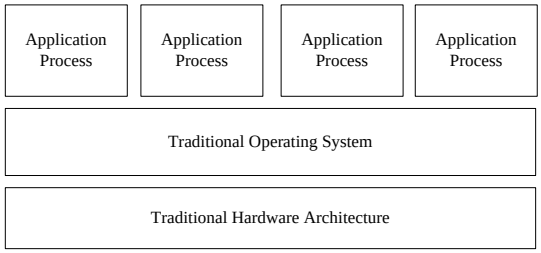

Figure 3. System Architecture of Compilation Approach

Traditional computing system does not support object-oriented programming. For an application that is written in object-oriented programming languages like C++, it is compiled into executable file for execution. An application process will be created for the execution of the program. Different applications execute in their own addressing spaces, and they are invisible from one and other by the use of virtual memory system [7]. In traditional hardware, protection mechanism normally implemented with page or segment table where access right information is maintained, as shown in Figure 4.

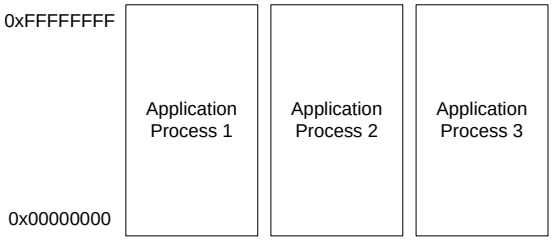

Figure 4. Addressing Space of Compilation Approach

- Pros

The compiler takes advantages of the processor architecture for compiling object-oriented programming language. With compilation, software is optimized with the features supported in nowadays hardware.

- Cons

A compiled executable program may not be necessarily manipulating objects. Objects may be compiled or translated into subroutines, which is machine readable, for direct execution. Object-oriented programming features may be removed during compilation. E.g. compilation removes the dynamicity behavior of object-oriented programming. Modification of a single class requires the whole application to be recompiled. Besides, the overhead of context switching in multitasking environment increases because of the need of page table updating.

V. VIRTUAL MACHINE APPROACH

In the virtual machine approach [8], the system consists of a traditional processor, a traditional operating system, some native application process, and an object virtual machine process, as shown in Figure 5. Similar to the compilation approach just mentioned in above section, native application process and the object virtual machine process are running in separated memory spaces.

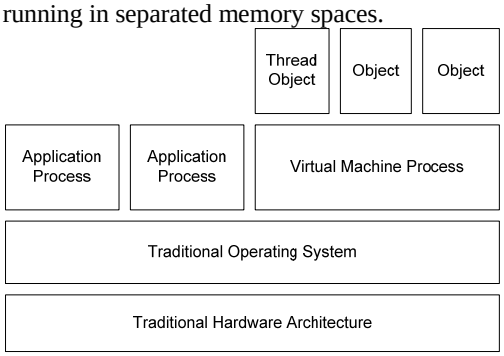

Figure 5. System Architecture of Virtual Machine Approach

To support object technology in this approach, an object virtual machine application is built on top of the traditional operating system. In the view of the operating system, it is just an application process like others. Protection of different processes is the same as in compilation approach. The addressing space of this approach is shown in Figure 6.

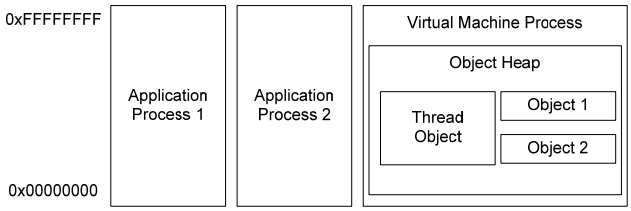

Figure 6. Addressing Space of Virtual Machine Approach

Objects are manipulated on the virtual machine. The virtual machine provides functionalities for newing objects, object communications, dynamic object linking, class loading, etc. Objects are allocated inside the heap, which resides in the same addressing space of the virtual machine process. The heap contains all the objects in the system. Protections of objects are managed by the virtual machine through software emulation. Multitasking of the object computing system is implemented by threading, where thread is inherited from object. Threads can be implemented by software emulation or mapped to system native threads. It depends on the implementation of the virtual machine.

- Pros

Without the modification of a hardware platform and current system, object technology can be supported through software emulation. It provides flexibility to the current systems for supporting objects instead of compiling objects.

- Cons

The virtual machine approach is based on software emulation. Two layers of software, virtual machine and operating system, introduce much overhead to the object computing system. Besides, traditional hardware does not manipulate object. Software emulation routines are required for the manipulation of objects, which increases the memory footprint of the system.

VI. Object-Oriented Hardware Approach

In the object-oriented hardware approach [4,5,6], it consists of an object-oriented processor and an object-oriented operating system. Object runs on the object-oriented operating system and shares the same heap, i.e. the same addressing space. The operating system manages the heap and sees only objects in the system. The architecture of object-oriented hardware is shown in Figure 7.

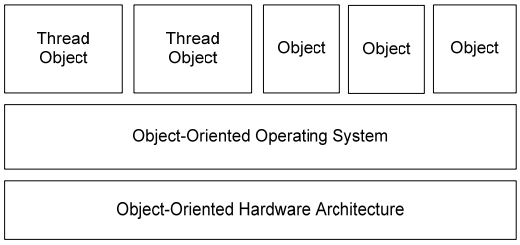

Figure 7. System Architecture of Object-Oriented Hardware Approach

In the point of view of the object computing system, operating system sees all the addressing spaces. There are no processes in an object computing system but tasks. Multitasking is achieved by threading. Thread objects is created for each thread in the system to maintain the status of the thread. In object computing system, protection of objects is governed by the object-oriented operating system with the protection features offered by the object-oriented processor. Different object-oriented processors may offer different features for object manipulation and protection. Object-oriented operating systems are designed to use these features for object computing support. The addressing space of hardware approach is shown in Figure 8.

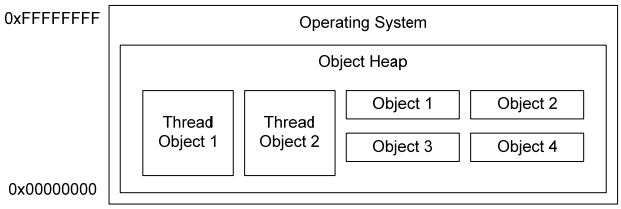

Figure 8. Addressing Space of Object-Oriented Hardware Approach

- Pros

This approach provides a pure object-oriented computing system for which everything is inherited from object. Object manipulation becomes more direct, with the aid of object-oriented features provided by the object-oriented processor. The efficiency of the object computing system is thus increased. For multitasking support, threading in object computing system reduces the overhead upon context switching since it requires no page table updating.

- Cons

This approach requires an object-oriented processor and operating system. Very few of them use in the market. Effort to implement such a system is large. Besides, migration to a new object-oriented operating system removes the compatibility. Software application developed with traditional system cannot be executed on the new object computing system. Extra efforts are required for the software developer to port their applications to the new system.

VII. Conclusion

Three different approaches for supporting object-oriented computing system have been discussed and compared. They are Compilation, Virtual Machine, and Object-oriented hardware approach. Pros and cons of these three approaches are discussed. Hardware architecture supports object manipulation more directly, thus increasing the efficiency of the object computing. Although the hardware approach is hardware-consuming, as the rapid development of silicon technology and billions of transistors can be integrated in a single chip, hardware request will be easy to meet. Hardware supporting directly object-oriented programming will become a promising approach.

Acknowledgment

The work described in this paper was supported by a fund from Yuexiu District Science, Technology and Information Bureau, Guangzhou (No. 2011-GX-042). It was also supported by funds from Guangdong Provincial of Science and Technology (No. 2011B090400408 and 2011B090400621).

References

[1] Weisfeld, Matt. The Object-Oriented Thought Process, Third Edition. Addison-Wesley. ISBN 0-672-33016-4, 2009

[2] John L. Hennessy, David A. Patterson, Computer Architecture, Fifth Edition: A Quantitative Approach, The Morgan Kaufmann Series in Computer Architecture and Design, Sep. 2011

[3] Craig, Iain, “The Interpretation of Object-Oriented Programming Languages”, Springer, 1999, ISBN 1-85233-159-3

[4] Hangal, Sudheendra; O'Connor, J. Michael, "Performance analysis and validation of the picoJava processor." IEEE Micro, Volume 19, Issue 3, May 1999

[5] Yijun Liu, Anthony S. Fong, Fangyang Shen: HISC: A computer architecture using operand descriptor. Computers & Electrical Engineering 38(3): 746-755, 2012

[6] Schoeberl, M. A Java processor architecture for embedded real-time systems. Journal of Systems Architecture 54 (1–2): 265–286. 2008

[7] B. Furht, V. Milutinovic, “Microprocessor Architectures for Virtual Memory Management”, Computer Architecture Tutorial, The Computer Society of The IEEE, 1987, page 191-209, ISBN 0-8186-0704-1.

[8] Venners, Bill, “Inside the Java 2 Virtual Machine”, Second Edition , McGraw Hill, 1999, ISBN 0-07-135093-4.