International Journal of Latest Engineering and Management Research (IJLEMR)
ISSN: 2455-4847
www.ijlemr.com ‖ Volume 1 - Issue 8 ‖ September 2016 ‖ PP. 29-35

Exploring the Computational Capabilities of Object Oriented Programming Languages

[1]Onu Fergus U., [2]Asogwa Samuel C.
[1]*Computer Science Department, Ebonyi State University, Abakaliki – Nigeria*
[2]*Computer Science Department, Michael Okpara University of Agriculture, Umudike- Nigeria*
Corespondence author: uche.fergus@gmail.com

Abstract: The real reason for the deployment of computers into the activities of man is to ease computational burdens especially complex computations. This ability to carry out complex computations by computers is built into them through software systems. Most modern software systems are developed with object oriented programming (OOP) Languages and tools. This paper takes an explorative look into the features of object oriented programming languages (OOPLs) in order to discover and expose their complex computational potentials. We state that the computational capabilities of OOPLs come from their in-built features such as data abstraction, polymorphism, encapsulation and inheritance. The paper used sample codes for handling complex computational problems in two OOPLs namely java and C++ to demonstrate these computational capabilities. It was discovered that these capabilities were also applied in the implementation of several software systems/applications in artificial intelligence (fuzzy logic, knowledge representation models etc), complex technical computations and in building interactive communities of computationally augmented objects.

Keywords: Complex Computation, Software Systems, Object Oriented Programming Languages, Data Encapsulation, Knowledge Representation

I. Introduction

Programming is a process of writing instructions to the computer hardware with the intension of solving human and societal problems. There are several approaches to writing computer programmes. These approaches as established by the various computer language vendors are called, programming languages. A programming language is the language the computer understands, obeys and, through which computer instructions are written. When a computer instruction is written in a logical pattern understandable by the computer it is called a computer program. When several of these programs that solve problems are combined as a unit we call it software or an application. It is these programmes and the applications/softwares that drives or controls the hardware. Software that solves specific tasks (computations) is called application software, while those that control and manage the computer hardware are known as systems software or operating systems. There is the need to understand the computational abilities inherent in programming languages. This work concentrates in the exposition of object oriented programming (OOP). This paper will help the reader appreciate the features of OOP and equally understand why OOP is suitable in certain programming style in implementing mega programming projects.

II. Overview of Programming language Paradigms

Paradigm" (a Greek word meaning example) is commonly used to refer to a category of entities that share a common characteristic.
We can distinguish between three different kinds of Software Paradigms:

- ***Programming Paradigm*** is a model of how programmers coomunicate an calculation to computers
- ***Software Design Paradigm*** is a model for implementing a group of applications sharing common properties
- ***Software Development Paradigm*** is often referred to as Software Engineering, may be seen as a management model for implementing big software projects using engineering principles.
- **Programming Paradigm**

A Programming Paradigm is a model for a class of Programming Languages that share a set of common characteristics. A paradigm is not a type of programming language. It is simply an approach to programming that is suitable for solving problems through a programming language. Common programming paradigms can be classified as, procedural (imperative), logic, object oriented, etc. [1, 6] Table 1. shows a brief of the differences that exist among the programming paradigms

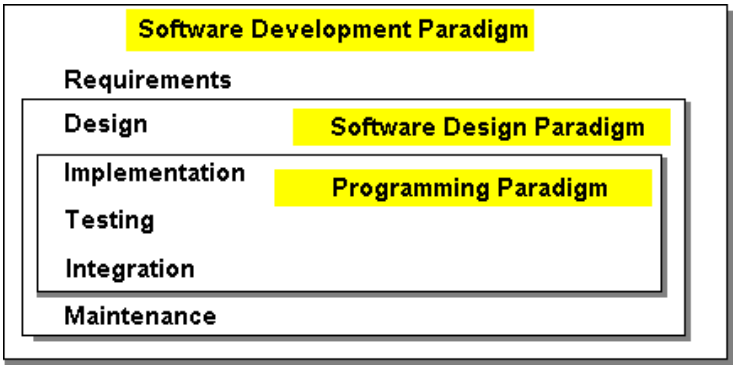

Fig. 1. Software Development Paradigm in Software life Cycle [1]

Table 1. Shows a brief of the differences that exist among the programming paradigms[2,3]

	Paradigm	**Features**
1	Procedural	Expresses the procedure to be followed to solve a task. Make use of loops. Has side effect such as; functions modifying variables
2	Functional	Makes use of recursive functions. However, it makes recursive calls and changes the parameters of those calls.
3	Object Oriented Programming (OOP)	Object-oriented programming views the world as a collection of objects that have internal data and external means of accessing parts of that data. The goal of object-oriented programming is to think about the problem by dividing it into a collection of objects that provide services that can be used to solve a particular problem. One of the main tenets of object oriented programming is encapsulation -- that everything an object will need should be inside the object. Object-oriented programming also emphasizes reusability through inheritance and the ability to extend current implementations without having to change a great deal of code by using polymorphism.[2]. Objects interact by means of message passing .Objects in classes are similar enough to allow programming of the classes, as opposed to programming of the individual objects. Classes are organized in inheritance hierarchies[3]
4	Logic Programming	(i)Automatic proofs within artificial intelligence. (ii) Based on axioms, inference and rules. (iii) program execution becomes a systematic search in a set of facts, making use of a set of inference[3]

III. Computational Capabilities of OOP

The computational Capabilities of object oriented programming are best understood through their characteristics. The major characteristics of object oriented programming are; (A) data abstraction, (B) encapsulation, (C) Polymorphism and (D) Inheritance.

(A) Data Abstraction

A class defines a new data type, use **private** instance variables to hide the choice of representation. **private** declarations are only visible inside the class. Data abstraction provide sufficient public methods to the outside world to play with the functionality of the object and to manipulate object data, i.e., state without actually knowing how class has been implemented internally [4,7]

Advantages of data abstraction are; (i) Client don't need to know about representation. (ii) Localized impact of changes. The disadvantages of data abstraction are; (i) More code to write and maintain (ii) Run-time overhead (time to call method) [4]

```
public class StringSet {
  // OVERVIEW: StringSets are unbounded, mutable sets of Strings.
  //    A typical StringSet is {x1, ..., xn}
  // Representation:
  private Vector rep;
  public StringSet () {
    // EFFECTS: Initializes this to be empty: { }
    rep = new Vector ();
  }
  public void insert (String s) {
    // MODIFIES: this
    // EFFECTS: Adds s to the elements of this:
    //     this_post = this_pre U { s }
    rep.add (s);
  }
```

Code example of data Abstraction in java
Fig 2. Implementation of StringSet [4]

(B) Polymorphism

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.
Any Java object that can pass more than one IS-A test is considered to be polymorphic. In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.
The only possible way to access an object is through a reference variable. Once declared, the type of a reference variable cannot be changed. [5]

```
public interface Vegetarian{}
public class Animal{}
public class Deer extends Animal implements Vegetarian{}
Deer d = new Deer();
Animal a = d;
Vegetarian v = d;
Object o = d;
```

// All the reference variables d, a, v, o refer to the same Deer object in the heap.
Fig. 4 A Short Java code Example of Polymorphism [5]

(C) Data Encapsulation

Encapsulation is an Object Oriented Programming concept that binds together the data and functions that manipulate the data, and that keeps both safe from outside interference and misuse. Data encapsulation led to the important OOP concept of data hiding. Data encapsulation is a mechanism of bundling the data, and the functions that use them and data abstraction is a mechanism of exposing only the interfaces and hiding the implementation details from the user. C++ supports the properties of encapsulation and data hiding through the creation of userdefined types, called classes.

(i) Genuine Object Encapsulation

Researcher in [11] used encapsulation in the implementation of his genuine object oriented programming (GOOP) - an integrated physical and computational construction kit in which children can use new “Things That Think” technology to build interactive communities of computationally augmented objects. GOOP provides an environment for children to explore and expand their own “Theories of Mind”, allowing them to construct powerful ideas about the nature of metaphor and shared understandings. He posits that, the mechanism that drives a genuine object’s function can consist of both computational and physical (e.g., motors and gears) elements. GOOP’s ability to encapsulate these multiple dimensions in a physical object is patterned on OOP’s ability to encapsulate computational procedures and data in an easily accessible software object.

```
#include <iostream>
Using namespace std;
      Class Adder {
            Public;
      // Constructor
      Adder ( int I = 0) {
                  Total = I }
          // interface to outside world
          Void addNum (int number) {
          Total += number; }
          // interface to outside world
          Int getTotal(){
          Return total ;
          };
          Private
          // hidden data from outside world
          Int total;
          };
          Int main() {
          Adder a;
          a.addNum(10);
          a.addNum(20);
          a.addNum(30);
          cout<<    "Total    <<a.getTotal()
<<endln;
          return 0;  }
```

Fig. 4 Data encapsulation implementation in C++. [7]

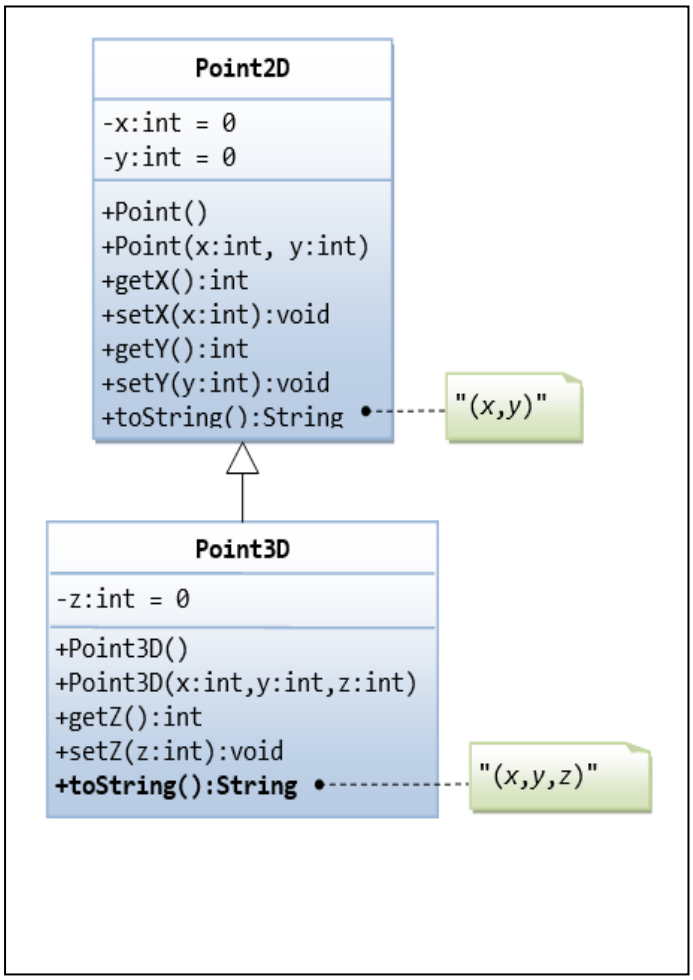

Fig. 5 **Inheritance**; The Point2D and Point3D Classes [9]

(D) Inheritance

Inheritance is a principle that allows you to apply your knowledge of a general category to more-specific objects. In Java, inheritance is a mechanism that enables one class to inherit both the behavior and the artributes of another class [8]

In OOP, we often organize classes in *hierarchy* to *avoid duplication and reduce redundancy*. The classes in the lower hierarchy inherit all the variables (static attributes) and methods (dynamic behaviors) from the higher hierarchies. A class in the lower hierarchy is called a *subclass* (or *derived*, *child*, *extended class*). A class in the upper hierarchy is called a *superclass* (or *base*, *parent class*). By pulling out all the common variables and methods into the superclasses, and leave the specialized variables and methods in the subclasses, *redundancy* can be greatly reduced or eliminated as these common variables and methods do not need to be repeated in all the subclasses. For example, A subclass inherits all the variables and methods from its superclasses, including its immediate parent as well as all the ancestors. It is important to note that a subclass is not a "subset" of a superclass. In contrast, subclass is a "superset" of a superclass. It is because a subclass inherits all the variables and methods of the superclass; in addition, it extends the superclass by providing more variables and methods. [9]

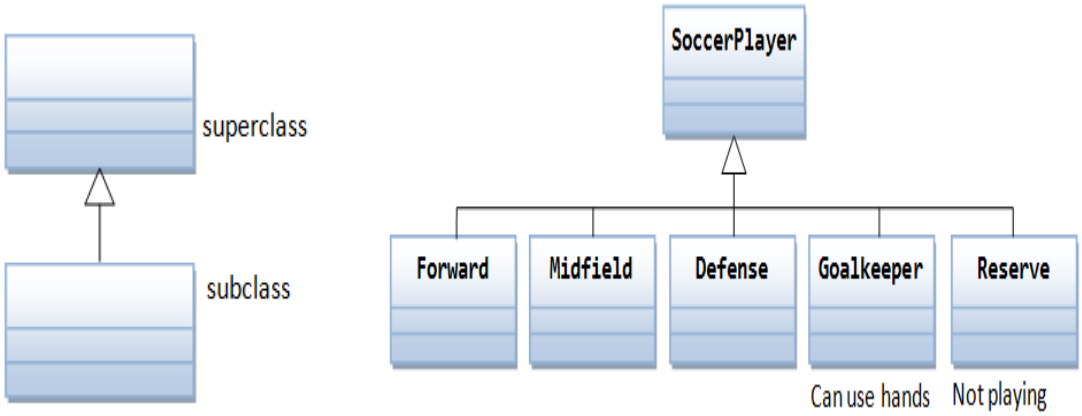

Fig. 4 Illustrations of inheritance in OOP [9]

(E) Inheritance in Object-Oriented Knowledge Representation

Nowadays the design and development of knowledge-based systems for solving problems in different domains are important tasks within area of artificial intelligence. Currently there are many different knowledge representation models (KRM), the most famous of which are logical models, production models, semantic networks, frames, scripts, conceptual graphs, ontologies, etc. All of these KRMs have their own specifics and allow representing of some types of knowledge. However, the certain Programming paradigm should be chosen for implementation of any particular KRM. For today the most famous and commonly used programming paradigm is an object-oriented programming (OOP). It gives us an opportunity of efficient implementation of many existing KRM.

within fuzzy frames slots can contain fuzzy sets as values. Secondly, the inheritance through is-a slot can be partial. Such extension of frames allows describing of objects and classes which have partial properties, i.e. properties which inherent with some measure. It means that such properties are not strictly true or false for the object or class. This kind of inheritance is called weaker inheritance. [10]

(F) OOP in Complex Technical Computing Applications

Object-oriented programming is a formal programming approach that combines data and associated actions (methods) into logical structures (objects). This approach improves the ability to manage software complexity—particularly important when developing and maintaining large applications and data structures. The object-oriented programming capabilities of the MATLAB® language enable you to develop complex technical computing applications faster than with other languages, such as C++, C#, and Java™. You can define classes and apply standard object-oriented design patterns in MATLAB that enable code reuse, inheritance, encapsulation, and reference behavior without engaging in the low-level housekeeping tasks required by other languages. Object-oriented programming in MATLAB involves using:

(i)Class definition files, enabling definition of properties, methods, and events
(ii)Classes with reference behavior, aiding the creation of data structures such as linked lists
(iii)Events and listeners, allowing the monitoring of object property changes and actions [12]

(G) Utility of Object-oriented Programming in Complex System Modeling

Researchers in [14] posit that with OOP, a complex system can be broken down into simple independent components, called objects. Objects individually display simple behavior. Simulations are configured at run-time by selecting appropriate objects. Complex system behavior is a result of message passing (interac-tion) among objects. Object-oriented programming languages possess certain characteristics that enable rapid prototyping of models with minimal software maintenance. The server prompts the user for objects to be integrated (Figure 5). The user makes a selection, causing the server to send an initialization message to the appropriate object. The receiving object interprets the initialization message and, if necessary, queries the user for more information. For example, the user may "tell" the server to simulate crop growth. The server then sends an initialization message to Crop which, in turn, queries for the type of crop to be simulated

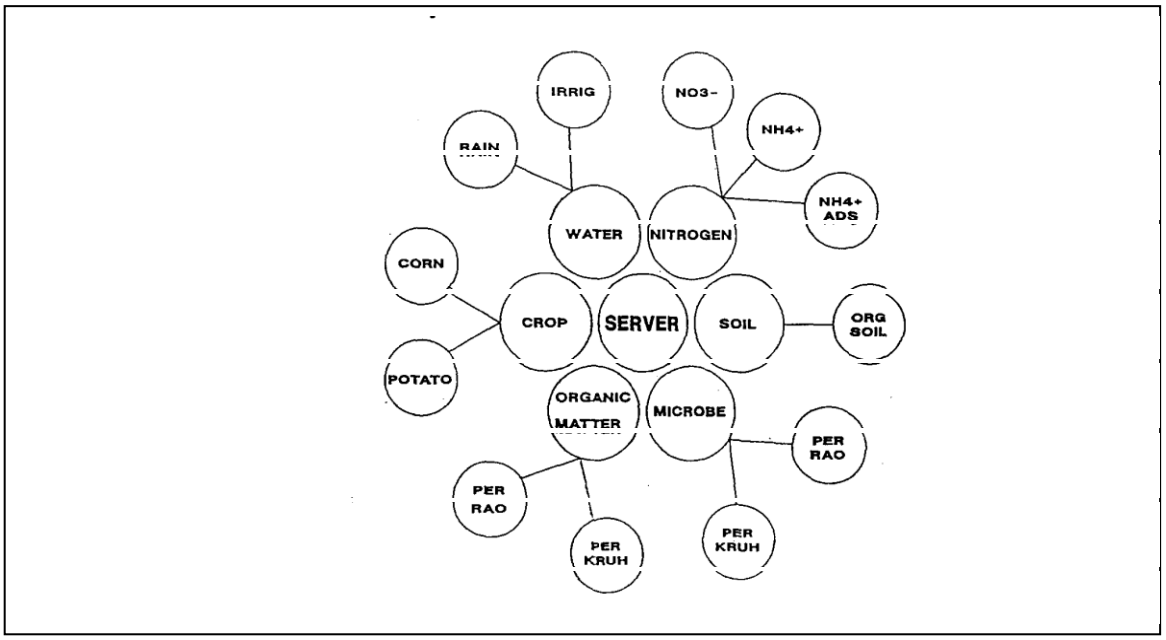

Fig 5. Message sending. A server object acts as a controller, integrating the selected objects to form the desired simulation. The server initiates communication with the first tier of subclass objects which, in turn, send messages to their subclass objects.[14]

(H) Parameterize Urban Design Codes With BIM and Object-oriented Programming

Researchers in [15] used OOP and building information modeling (BIM) to calculate the Floor Area Ratio (F.A.R.), a ratio of the total floor area of the buildings to the property area. To obtain this value from the parametric urban code model, two values were used; gross floor area and property area. The gross floor area can be read from the building object and the property area can be read from the parcel object. Another example is the maximum building footprint area. In general, a certain ratio of the property area is allowed for the building footprint area. To do so, the property area and the allowed ratio need to be collected from the parcel object, and it can be transferred to the building object. They also used complex datasets and tools such as geographic information systems data modeling urban design codes to model open spaces, streets, buildings, definition of parameter lists and code models

(I) Calculating Heading in 2D Games.

Trigonometric functions like sine (sin) and cosine (cos) are often used in game programming for a number of tasks. For example, rotating a spaceship or determining horizontal and vertical velocity based upon the angle the character/ball/missile is moving, moving a non-player-character in a more pleasing way than simply left, right, up, down or diagonal (think about the deadly swooping of a Galaxians enemy). And what about a ripple effect; perhaps an explosion or pebble-splash, starting from a centre point and radiating outwards at an increasing radius.

The heading of two dimensional games can be calculated using trigonometric functions. The programming languages and application programming interface (API's) provides the complex math wrapped up in a simple method/function call.

For example in Java can be used to write code to control the horizontal and vertical velocity of say a spaceship to make sure it travelled in precisely the same direction that it was facing. See fig. 6

```
// facingAngle can be any angle between 1 and 360 degrees
// the Math.toRadians method simply converts the more conventional
// degree measurements to radians which are required by the cos and
sin methods.
horizontalVelocity = speed * Math.cos(Math.toRadians(facingAngle));
verticalVelocity = speed * Math.sin(Math.toRadians(facingAngle));
```

Fig. 6. Java sample code that can control the vertical and horizontal velocity of object [16]

IV. Conclusion

In this paper, we have studied some of the computational capabilities of OOP and discovered that these abilities are derived from their characteristic features such as data abstraction, polymorphism, encapsulation and inheritance etc. In exploring the computational abilities of OOP we discovered that application of these features are applied in artificial intelligence such as in knowledge representation models, fuzzy logic and in development of complex scientific applications such as in modeling tolls. Etc. We have also given some useful code examples and models that will be of great guide to the programmer.

Reference

[1.] Software Paradigms (Lesson 1): "Introduction & Procedural Programming Paradigm"
[2.] *Eric Suh* "The Tower of Babel -- A Comparison Programming Languages retrieved from: http://www.cprogramming.com/langs.html
[3.] Overview of the four main programming paradigms. Retrieved from: http://people.cs.aau.dk/~normark/prog3-03/html/notes/paradigms_themes-paradigm-overview-section.html
[4.] David Evans (2003) Lecture 5: "Implementing Data Abstractions". Retrieved from http://www.cs.virginia.edu/cs201j/lectures/lecture5.ppt
[5.] Polymorphism. Retrieved from; http://cecs.wright.edu/~tkprasad/courses/cs480/L3OOP.pdf
[6.] Onu F. U, Asogwa S. C and , Ugwoke F.N: "comparative analysis of programming languages - a study" IJCSIT, Vol. 3, Issue 4 (Aug, 2016)
[7.] Data Abstraction in C++. Retrieved from: http://www.tutorialspoint.com/cplusplus/pdf/cpp_data_abstraction.pdf
[8.] Joyce Farrel. "java programming, second edition thomson course technology 2003
[9.] Inheritance; retrieved from http://www3.ntu.edu.sg/home/ehchua/programming/java/j3b_oopinheritancepolymorphism.html
[10.] Dmytro Terletskyi; "Inheritance in Object-Oriented Knowledge Representation" arXiv:1510.04212v1[cs.AI] 14 Oct. 2015
[11.] Richard Daniel Borovoy: (1996) "Genuine Object Oriented Programming" Massachusettes Institute of Technology.
[12.] "Exploring the computational capabilities of oop" Retrieved from; http://www.mathworks.com/discovery/object-oriented- programming.html? requested Domain=www.mathworks.com
[13.] Stuart McGarrity "Introduction to Object-Oriented Programming in MATLAB". Retrieved from: http://www.mathworks.com/company/newsletters/articles/introduction-to-object-oriented-programming-in-matlab.html
[14.] William M. Clapham and Carol J. Crosby(1992) "Utility of Object-oriented Programming in Complex System Modeling" *Mathl. Comput. Modelling* Vol. 16, No. *617*, pp. *45-50, 1992*
[15.] Jong Bum KIM, Mark J. Clayton and Wei YAN "Parameterize Urban Design Codes with BIM And Object-Oriented Programming" Open Systems: Proceedings of the 18th International Conference on Computer-Aided Architectural Design Research in Asia (CAADRIA 2013), 33–42.
[16.] "Calculating Heading in 2D Games." Retrieved from: http://gamecodeschool.com/essentials/calculating-heading-in-2d-games-using-trigonometric-functions-part-1/