Open Access Library Journal
2017, Volume 4, e3926
ISSN Online: 2333-9721
ISSN Print: 2333-9705

Harmonic Paradigm

Yoosef Habibi

Department of Computer, Faculty of Computer Engineering and Science, Shahid Beheshti University, Tehran, Iran
Email: yoosef.habibi1@gmail.com

How to cite this paper: Habibi, Y. (2017) Harmonic Paradigm. *Open Access Library Journal*, **4**: e3926.
https://doi.org/10.4236/oalib.1103926

Received: September 7, 2017
Accepted: October 21, 2017
Published: October 24, 2017

Abstract

Programming paradigms are a way to classify programming languages according to the style of computer programming [1]. In this paper, we tried to present a new programming paradigm by interweaving the concepts of harmony and an inspiring sample of nature (jellyfish and its way of life). This model includes some of the advantages of available paradigms such as object-oriented and functional paradigm as well.

Subject Areas

Computer Engineering

Keywords

Programming Paradigm, Harmonic, Harmonic Paradigm, Object-Oriented Programming, Functional Paradigm

1. Introduction

A programming paradigm is a style or "way" of programming [2].

In the past 40 years, many paradigms were developed, paradigms like functional and object-oriented.

Every paradigm has an abstraction view considering data, operands, and control and generally everything from the creation of a program to its execution.

Properties and techniques that are used by each paradigm, implies different application area. For instance, every time we have programs with complicated data and interface it is better to use object-oriented paradigm. Generally, what makes the difference between paradigms in addition to the concept of programs and its execution, are the concept of variables [3].

Some of the well-known paradigms:

- Imperative,
- Procedural,

- Functional,
- Object oriented,
- Event driven,
- Flow driven [2].

Imperative program paradigm is one of the oldest programming paradigms that evolved from the assembly language and its properties are a reflex of John Von Neuman's computer architecture laws. This paradigm consists of explicit commands and call the procedure, each procedure affects some data.

Functional paradigm is an old paradigm as well. The program divided into sets of functions; each does a certain part of the program [3]. The fundamental elements of functional programming are variables and functions [5] [10].

Logic paradigm is a set of rules and facts that specify object properties and using the inferences obtained from the available phrases, it starts building the intended objects.

Object oriented paradigm, the program described using objects and interactions among them [3]. Objects are modular units consisting data and operations on data .These modular units-objects-communicate through specified interfaces [5] [10].

Objects communicate with each other and together build the whole program. Objects are instances of the classes. Every class represents a set of objects with similar attributes and operations [3]. One of the main features of oop—object oriented programming—is having encapsulation. Encapsulation is a technique for breaking programs into pieces with specified external interfaces. Oop supports two ways of encapsulation: Object encapsulation and Class encapsulation.

Object encapsulation ensures the accessibility of private members within the object. Class encapsulation provides the accessibility of private members by objects of same class [6].

This paper illustrates a new programming paradigm—Harmonic Paradigm—with its specific features. In the following paper, it will be discussed the comparison between Harmonic Paradigm and mentioned paradigms.

2. Explanation of Harmonic Paradigm

Definition of Harmony:

- A pleasing arrangement of parts
- An interweaving of different accounts into a single narrative [4].

As an example of a set that has harmony-what we would call harmonic set-consider a bike in which arranging the wheels, chains and the other parts of a bike in order to make it suitable for riding makes the bike a good example of a harmonic set.

Suppose a part of seawater as a harmonic system that has some of the following components:

Sea creatures (like jellyfish that only has an entry and exit) and sea flows.

These flows lead the jellyfishes to move and the only thing that Jellyfishes do is letting the food to enter and exit. Jellyfishes live in this way.

Presenting the definition of the harmony and the sample of nature is for the sake of discussing how the harmonic paradigm is concluded from interweaving these materials.

The presented paradigm in this paper with the name of harmonic paradigm has three fundamental components:

1) Components which are merely acceptor. (Like the jellyfish)

2) Components which implement the general act of the program (like the sea flows) that is named action profile.

3) Functional components which implement the influence of the being components in the program (like jellyfish and flows) in order to both save the side effects of the components and stabilize the system and is named self action profile.

Furthermore, we are going to exactly describe each of these three basic components:

1) Action profile

It is a kind of structure which has a set of functions which it is presented specific order in itself which works as a functional component and accepts some input and at last it produces an output and this is only thing that is visible from outside and the number or the kind of the functions aren't visible. In other words, every action profile has encapsulation to the others.

Generally, every action profile implements a fundamental action in a program.

The general structure is like this:

```
Action profile A (inputs){
A1 ();
A2 (); // the order in which A call its subroutines
A3 ();
A1 () {implementation}
A2 () {implementation}
A3 () {implementation}
} // end of action profile A
```

2) Self action profile

The self action profiles are functioned similarly to action profiles, but they are different from action profiles naturally. The action profiles implement fundamental actions of the program; on the other hand, self action profiles are a set of functions which for these aims:

a) Implementing the behavior and side effects.

b) Considering a condition for the stability of the system.

Generally, they are called implicitly and their priority is more than action profiles'.

Fundamental actions: a set of actions in the system that in case of being omitted, the main functional feature and the facilities of the system will be reduced.

According to this definition, omitting the self action profiles doesn't reduce the fundamental actions and it was mentioned it was a set of rules for environ-

mental conditions (program) that is implemented and naturally in case of omitting them, there will be a decrease in the implementation.

3) Acceptor

Acceptors are components of the program which are working together; same as objects in object oriented paradigm with only one great difference that in the object oriented paradigm, every object is empowered, because it can do some actions; but in this presented point of view, components are not empowered and the defined actions in the system, depending on conditions, involve the acceptors in conditions or don't.

For using the acceptors by action profiles and self action profiles, these approaches are available:

a) Table approach

b) Exclusive lock approach

Table approach:

In this approach, a table is used that the permission of the acceptors to reach the action profiles (Figure 1) and self action profiles is mentioned like the following shaped:

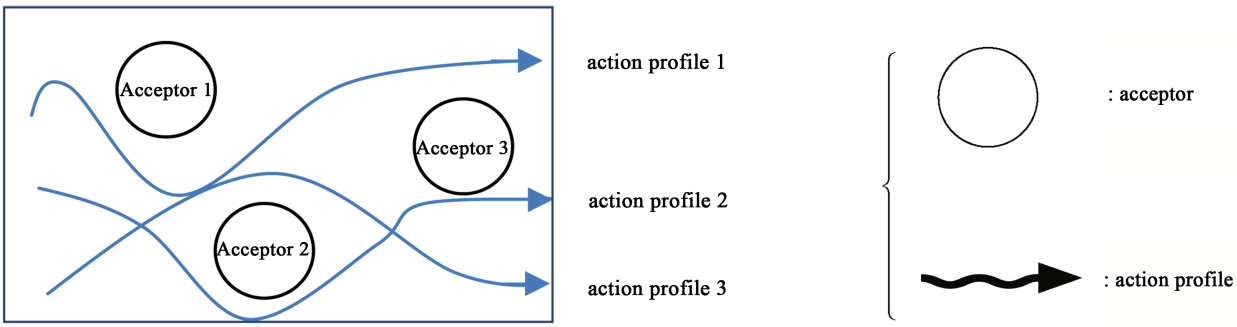
Disadvantage: whenever the number of (Σ|acceptors| × Σ|action profiles + self action profiles|) increases, then in this way there will be a big table that must be stored in the memory and its initialization should be run in O (Σ|acceptors| × Σ|action profiles + self action profiles|).

Exclusive lock approach:

In this approach, any acceptor has a parameter called exclusive lock (Figure 2).

This approach has two attitudes:

i) Specifying just one function with the name of control exclusive lock which has the duty to initialize the exclusive lock parameter to any acceptor and in this



action profile / acceptor	action profile 1	action profile 2	action profile 3
acceptor 1	Y	N	N
acceptor 2	N	Y	N
acceptor 3	N	Y	N

Y: (*Yes*) *has access*
N: (*No*) *doesn't have access*

Figure 1. Table approach sample.

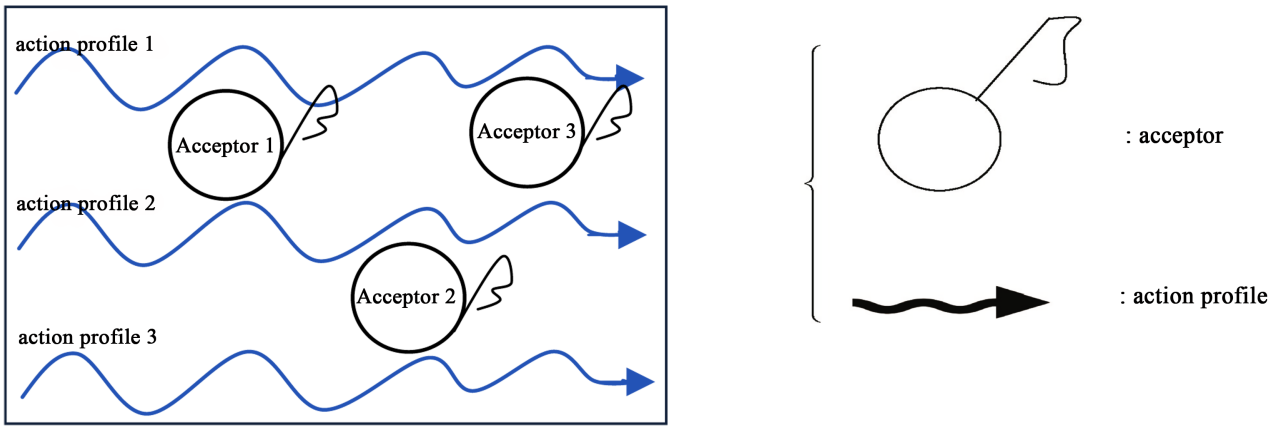


Figure 2. Exclusive lock approach.

attitude the acceptor has the ability to control. In addition, we have somehow provided encapsulation; because this attitude lets any acceptor to decide which action can access to it. For example, in a university automation system, the student decides whether the check-marks function can have access to it or not.

ii) In this attitude, instead of giving the control duty to any acceptor, there is a global control with the name of a specific self action profile and whenever the exclusive lock parameter of an acceptor needs to change, that specific self action profile set the exclusive lock parameter for that requested acceptor. Even in the table attitude, none of the action profile and self action profile has permission to change the table and a specific self action profile does this action instead.

Eventually, we would like to mention the fact that choosing among these methods depend on programmers and circumstances in which they are programming with.

Eventually, it is necessary to mention that all these three fundamental components (acceptor, action profile, self action profile) are implemented by classes. In other words, we have three kinds of classes which implement one of the main program's characteristic:

Acceptor as a class that defines the components that are communicating with each other, action profile as a class which define a flow that models fundamental action and self action profile defines permissions, access level and environmental communicating actions between different parts of the program. There is more than one self action profile class and each class focuses on different parts of components of the program and their relations.

Benefits and specific features:

1) Harmonic paradigm has encapsulation. In this point of view, the program is divided into pieces (action profile classes, self action profile classes, and acceptor classes). Action profile and self action profile are black boxes that their implementations are not visible from outside of them. Acceptors are also hidden from action profiles but by self action profile's permission. And their accessibilities and communications are as following:

Action profile can change acceptor's variables with self action profile's permission.

This permission is defined by the programmer for programmer's preferences.

Self action profile can change acceptor's variables.

Self action profile can affect the performance of specific flows under certain conditions. These effects are defined by the programmer.

The result of flows and variables' state in acceptors can affect self action profile's state (Figure 3).

As an example, consider the following scenario:

Specific acceptor has some info that is needed by a specific flow.

Specific self action profile for this condition allows that flow to access the information.

After operating, flow makes an invalid value for a specific variable.

So, the state of that self action profile changes to be able to handle this issue.

2) Information hiding is a technique for determining which parts of a class should be visible to which clients. It's important because it affects design specifications. In other words, certain specifications should be visible whereas it doesn't cause hidden parts to expose [7].

In Harmonic Paradigm, action profile classes are only black boxes that perform an action and can use the result of each other. So, they are hidden from the rest of the program. Acceptors are hidden by default but by self action profile's permission. Permission can be defined by programmer's preferences.

3) In order to implement concurrent programming, it is necessary to consider a way for letting the processes communicate with each other. In order to do so, it is necessary to synchronize between asynchronous processes. Two common ways of synchronizing are:

The Mutual Exclusion that provides a critical section in the code for preventing writing into any shared variables while reading.

Conditional Synchronization delays a process for providing the requirements to get synchronized with other parts [8].

In order to support parallel programming, the concept of paradigm should support techniques that are used for parallel programming: partitioning and divide-and-conquer.

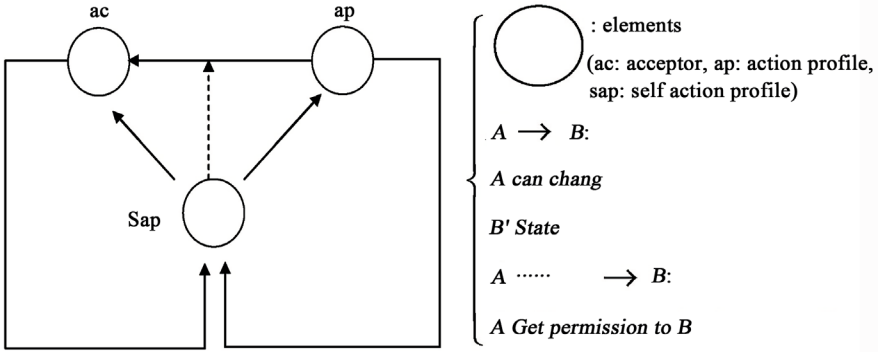


Figure 3. Elements and their effects on each other.

Partitioning: In this approach, the main problem is divided into smaller parts and these parts compute separately. There are two kinds of partitioning: Data partitioning and Functional partitioning.

In Data partitioning, the data with its functions is divided into parts and computes separately. In Functional partitioning, the main task is divided into smaller tasks and then they will be computed.

Mentioned techniques can use both message passing and shared memory for communication.

Divide-and-conquer: It recursively divides the program into sub-programs until the programs could not be divided into smaller parts. Eventually, it combines the results to solve the main problem [9].

Harmonic Paradigm is appropriate for both kinds of programming:

Action profiles as flows of actions can be computed separately on distributed systems or sequentially within a single processor. Self action profiles can define the circumstances for one single processor or can be defined for each processor individually and a small self action profile for communicating and controlling the whole program.

4) It has the advantages of the object oriented paradigm that its attitude is similar to the men's attitude.

5) In addition, due to the suitable hierarchy, implementation is divided into different levels which reduced the designing pressure on one part.

6) A better use of resources, not reaching a deadlock and remaining stable can be better implemented.

7) Generally, it uses the object oriented and functional paradigm advantages and it is individually better than those two paradigms.

8) All of the actions which are implemented in the program are studied in a specific layer; as a result, finding the errors and enhancing and optimizing the action of the system is much easier.

Simple example:

Suppose that we want to implement a university automation system with simple conditions by Harmonic paradigm, the only thing that we need to do is to design these following layers:

1) Acceptors: we need to implement the components that are interacting with each other in this level like:

Student {student's properties}

Staff {staff's properties}

Teacher {teacher's properties}

2) Action profiles: in this layer, we need to implement fundamental functions of the program like:

Checking recent status (students/staff/teacher)

Setting marks

Getting marks

Write a complaint

3) Self action profiles: in this part, we need to implement functions that determine the environmental conditions of the program like:

a) Permission to access parts/evidence.

b) Sending message to clients.

3. Conclusion

In this paper, we presented a model inspired by nature by presenting the concepts of harmony and programming paradigms and associating these two concepts with each other. This newly introduced model has much of the advantages of two common paradigms (object-oriented and functional paradigm). Besides, it has illustrated new advantageous features making it simpler to design and detect errors.

References

[1] https://en.wikipedia.org/wiki/Programming_paradigm

[2] http://cs.lmu.edu/~ray/notes/paradigms/

[3] Elena, B. (2005) Programming Paradigms in Computer Science Education. *International Journal of Information Theories & Applications*, **12**, 285-290.

[4] www.merriam-webster.com/dictionary/harmony

[5] Elmar, L. Object-Oriented Programming versus Functional Programming, a Comparison of Concepts, Special Topic in the Lecture on "Functional Programming with ML", University of Osnabrück, Winter Term 2001/02.

[6] Janina, V., Warwick, I. and Neville, C. Class Encapsulation and Object Encapsulation: An Empirical Study.

[7] Leavens, G.T. and Peter, M. Information Hiding and Visibility in Interface Specifications.

[8] Schneider, F.B. and Gregory, R. Andrews Concepts for Concurrent Programming.

[9] Wilkinson, B. and Allen, M. (2004) Parallel Programming Techniques & Applications Using Networked Workstations & Parallel Computers. 2nd Edition, Pearson Education Inc.

[10] Don, L. and Greg, W. (1995) Object-Oriented Programming and the Objective-C Language. NeXT Software Inc.