Amrita Vishwa Vidyapeetham

TIFAC-CORE in Cyber Security

20CYS312 - Principles of Programming Languages
Assignment-01: Exploring Programming Paradigms

Harshitha Ranjith

21st January, 2024

Introduction

A paradigm is a way of approaching a problem or carrying out a task. A programming paradigm is a way of approaching problems with programming languages, or with the tools and techniques at our disposal while adhering to a certain approach. There are many well-known programming languages, but when they are used, they all need to adhere to a technique or strategy, and this approach is known as paradigms. In addition to several programming languages, there are numerous paradigms to meet every need.

There are 2 types of programming paradigms:

1. Imperative Programming Paradigm
   (a) Procedural Programming paradigm
   (b) Object Oriented Programming Paradigm
   (c) Parallel Processing Approach
2. Declarative Programming Paradigm
   (a) Logic Programming Paradigm
   (b) Functional Programming
   (c) Database Processing Approach

1.Imperative Programming Paradigm

It is one of the oldest programming paradigms. It features a close relation to machine architecture and is based on Von Neumann architecture. It works by changing the program state through assignment statements, performing tasks step by step by changing the state. The main focus is on how to achieve the goal, consisting of several statements, and after execution, the result is stored.

Advantages:

- Very simple to implement
- Contains loops, variables, etc.

Disadvantages:

- Complex problems cannot be solved

- Less efficient and less productive
- Parallel programming is not possible

Example in C:

```
int marks[5] = { 12, 32, 45, 13, 19 };
int sum = 0;
float average = 0.0;
for (int i = 0; i < 5; i++) {
    sum = sum + marks[i];
}
average = sum / 5;
```

i)Procedural Programming Paradigm

It places a strong emphasis on procedure in relation to the underlying machine model. The imperative approach and the procedural approach are identical. Since the code could be reused, this feature was quite helpful when it was first put to use.

Examples: C ,C++ ,Java ,ColdFusion ,Pascal.

ii)Object Oriented Programming Paradigm

The program is written as a collection of classes and objects meant for communication. The smallest and basic entity is an object, and all kinds of computation are performed on the objects. More emphasis is on data rather than procedure. It can handle almost all kinds of real-life problems present today.

Advantages:

- Data security
- Inheritance
- Code reusability
- Flexible and abstraction is also present

Examples: Simula ,Java ,C++ ,Objective-C ,Visual Basic .NET ,Python ,Ruby ,Smalltalk

iii)Parallel Processing Approach

Processing computer instructions in parallel involves distributing them over several processors. The goal is to divide multiple processors so that a program can be executed more quickly. This strategy is similar to divide and conquer. Examples include NESL and C/C++ supported by some library functions.

2.Declarative Programming Paradigm:

It is divided as Logic, Functional, Database. In computer science the declarative programming is a style of building programs that expresses logic of computation without talking about its control flow. It often considers programs as theories of some logic.It may simplify writing parallel programs. The focus is on what needs to be done rather how it should be done basically emphasize on what code is actually doing. It just declares the result we want rather how it has be produced. This is the only difference between imperative (how to do) and declarative (what to do) programming paradigms. Getting into deeper we would see logic, functional and database.

i)Logic programming paradigms

It can be termed as abstract model of computation. It would solve logical problems like puzzles, series etc. In logic programming we have a knowledge base which we know before and along with the question and knowledge base which is given to machine, it produces result. In normal programming languages, such concept of knowledge base is not available but while using the concept of artificial intelligence, machine learning we have some models like Perception model which is using the same mechanism.

ii)Functional programming paradigms

The functional programming paradigms has its roots in mathematics and it is language independent. The key principle of this paradigms is executing a series of mathematical functions. The central model for the abstraction is the function which are meant for some specific computation and not the data structure. Data are loosely coupled to functions.The function hide their implementation. Function can be replaced with their values without changing the meaning of the program. Some of the languages like perl, javascript mostly uses this paradigm.

Examples of Functional programming paradigm: JavaScript ,Haskell ,Scala ,Erlang,Lisp,ML ,Clojure

iii)Database/Data Driven Programming technique:

The foundation of this programming technique is data and how it moves. Program statements are not hard-coded sequences of steps; instead, they are determined by data. A business information system's database application, which handles file creation, data entry, updates, queries, and reporting, is its central component. Numerous programming languages have been developed, mostly for use with databases. Take SQL, for instance. It can be used to filter, transform, aggregate (e.g., compute statistics), or invoke other programs on streams of structured data. Thus, it has a broad range of applications.

1 Paradigm 1: IMPERATIVE PARADIGM

What is Imperative Programming?

The imperative programming paradigm uses a sequence of statements to modify a program's state through the use of variables. The goal of the imperative paradigm is to specify how a program should execute through explicit instructions.

It's common to see imperative programming compared to declarative programming, which is the philosophy behind languages where the programmer specifies what to do, rather than how to do it.

If this is confusing, for now consider this: a cookbook instructs step-by-step how to cook a recipe, a restaurant menu shows what you can order, and you don't need to worry how it's made.

The imperative paradigm serves as the basis for later paradigms like the structured, procedural, and object-oriented paradigms.

Characteristics of Imperative:

The imperative paradigm defines variables and statements as the "tools on our utility belt" that we employ to construct practical programs. Let's discuss the imperative paradigm's features:

The key ingredients of the imperative paradigm are assignment variables and mutable variables. They are the ones who enable the intriguing and practical things to happen. They come together to form what is referred to as a program's state. After being initially set to a value, some variables have the capacity to change. These variables are referred to as changeable variables, whereas immutable variables are those whose values are fixed and cannot be altered. Take a look at the following program:

```
var first = 5
var second = 17
var result
result = first + second
print(result)
```

When we want a new variable within our program, we need to declare that variable. In our program, we've declared the variables first and second, and assigned their values to be 5, and 17 by using the assignment operator =. In common programming speech, we can say we've initialized a variable when we assign its value for the first time.

Language for Paradigm 1: OBJECTIVE C

It is an object-oriented, general purpose language and was created with the vision of providing small talk-style messaging to the C programming language. This language allows the users to define a protocol by declaring the classes and the data members can be made public, private and protected. This language was used at Apple for iOS and OS X operating systems. Swift language was developed at Apple in 2014 to replace this language. But still there are plenty of companies that are maintaining their legacy apps which are written in objective C.

The main difference in C and Objective C is that C is a procedure programming language which doesn't support the concepts of objects and classes and Objective C is Object-oriented language which contains the concept of both procedural and object-oriented programming languages.

Difference between C and objective c:

1) C is a procedure oriented programming language. Problems are solved step by step fashion. Whereas, Objective c is an object-oriented programming language. It adds syntax and semantics that allows for an object oriented language. But it doesn't support multiple inheritance property.

2) C language can be called the subset of Objective C. Whereas, Objective C can be called the super set of C language. It contains classes and objects in addition to C language.

3) The pointers used in C language are vulnerable to security attacks. Whereas objective C uses null pointers and hence is type safe compared to C.

4) It is basically a low level language that stands too close to assembly level language. Whereas, Objective C is a high-level language stuffed with small talk messaging style together with C.

5) C language doesn't incorporate any classes. Bjarne Stroustrup developed the C++ language with the main intent of adding object oriented features like class to the C language. Whereas, Objective C is object-oriented language and incorporates classes and offers dynamic runtime.

6) It follows the top-down programming approach.Whereas, Objective C follows the bottom-up programming approach.

7) In this language big program code is divided into small pieces of code which is called functions.Whereas in Objective C, big program code is divided into smaller codes which is called Objects and Classes.

8) It supports only pointers.Variables should be declared at the beginning of the program. Whereas in Objective C,it supports both pointers and references.In this language variable can be declared anywhere in the program.

Features of Objective c:

1. Objective C supports exception handling, which can be implemented using catch and try blocks.
2. It allows the definition of functions with default arguments.
3. Objective C provides a new operator for memory allocation and a delete operator for memory deallocation.
4. Using the concept of encapsulation, security can be achieved in Objective C.
5. It supports inline functions.
6. Objective C supports function and operator overloading.
7. It is known as an object-driven language.
8. Encapsulation,data hiding,inheritance, polymorphism, and abstraction are the key features.
9. Objective C supports templates.
10. It is well-suited for networking, gaming, etc.

Objective-C's Imperative Design :

Objective-C, while influenced by other paradigms, is primarily an imperative programming language. This means it focuses on giving instructions to the computer, telling it what to do step-by-step, rather than declaring what properties or relationships exist (like in declarative languages).

Let's see how some key features of Objective-C embody imperative principles:

1. Explicit Commands: Statements in Objective-C directly instruct the computer to perform actions. Every line of code, like function calls, assignment statements, and loops, tells the program what to do next.

2. Sequencing and Control Flow: Program execution follows a specific order dictated by the sequence of statements. Control flow structures like if statements, for loops, and while loops allow the program to make decisions and execute different blocks of code based on conditions.

3. Procedural Decomposition: Complex tasks are broken down into smaller, clearly defined functions or methods. This allows for code reuse, readability, and modularity.

4. State Management: Variables and objects hold the state of the program, representing data and memory locations. Imperative programs explicitly manipulate these states through assignments and method calls to achieve desired outcomes.

5. Side Effects: Many operations in Objective-C have side effects, meaning they change the state of the program outside the immediate context of the statement. This requires careful consideration of program state and the potential for unexpected behavior.

Paradigm 2: EVENT DRIVEN PARADIGM

What is Event Driven Programming?

Event Driven Programming is a programming approach that enables software to respond to events originating from external sources, such as user input or system changes. It allows the creation of dynamic applications where the flow of control is determined by the sequence of events, rather than a predetermined order of execution. The primary goal of Event Driven Programming is to make the software more responsive to user actions and to simplify the development process by providing a clear separation between event handling and other aspects of the software design.

Definition: Event Driven Programming: A programming paradigm that structures and organizes the flow of code around responding to events originating from external sources such as user input or system changes.

Key Components of Event Driven Programming: There are several key components in Event Driven Programming that work together to handle and process events in an efficient way. Understanding these components is essential for creating successful Event Driven applications.

Event Handlers: Event handlers are the backbone of Event Driven Programming. These are functions or methods that are designed to be triggered when a specific event occurs. For instance, when a user clicks on a button on a graphical user interface, an event handler associated with that button responds by executing the designated code.

Event handlers can be further categorized as:

i)Synchronous event handlers: Execute the code immediately when an event occurs. ii)Asynchronous event handlers: Allow other tasks to continue executing while the code for handling the event is being processed.

Example: A simple event handler in JavaScript for a button click event:

```
function displayMessage() {
  alert("Hello, World!");
}
```

Event Loop: The event loop is a continuous process that runs in the background and checks for any queued events. When an event is detected, the event loop dispatches it to the appropriate event handler for processing. It then moves on to the next event in the queue, ensuring that all events are handled as they occur. The event loop is responsible for managing the event queue and maintaining the responsiveness of the application.

Advantages of Event-Driven Programming:

- Flexibility: It is easier to alter sections of code as and when required.
- Suitability for graphical interfaces: It allows the user to select tools (like radio buttons etc.) directly from the toolbar.
- Programming simplicity: It supports predictive coding, which improves the programmer's coding experience.
- Easy to find natural dividing lines: Natural dividing lines for unit testing infrastructure are easy to come by.
- A good way to model systems: Useful method for modeling systems that must be asynchronous and reactive.
- Allows for more interactive programs: It enables more interactive programming. Event-driven programming is used in almost all recent GUI apps.
- Using hardware interrupts: It can be accomplished via hardware interrupts, lowering the computer's power consumption.

- Allows sensors and other hardware: It makes it simple for sensors and other hardware to communicate with software.

Disadvantages of Event-Driven Programming:

- Complex: Simple programs become unnecessarily complex.
- Less logical and obvious: The flow of the program is usually less logical and more obvious.
- Difficult to find error: Debugging an event-driven program is difficult.
- Confusing: Too many forms in a program might be confusing and/or frustrating for the programmer.
- Tight coupling: The event schema will be tightly coupled with the consumers of the schema.
- Blocking: Complex blocking of operations.

Relation between Event-Driven Programming and Object-Oriented Programming: We can combine Object-oriented Programming (OOP) and Event-driven programming (EDP) and use them together in the same code snippet.

When OOP is used with EDP:

All OOP fundamentals (encapsulation, inheritance, and polymorphism) are preserved. Objects get the ability to post-event notifications and subscribe to event notifications from other objects. How to distinguish between OOP with and without EDP: The control flow between objects is the distinction between OOP with and without EDP. On a method call in OOP without EDP, control flows from one object to another. The primary function of an object is to call the methods of other objects.

Language for Paradigm 2: NODE.JS

Node JS is an open-source and cross-platform runtime environment built on Chrome's V8 JavaScript engine for executing JavaScript code outside of a browser. It provides an event-driven, non-blocking (asynchronous) I/O and cross-platform runtime environment for building highly scalable server-side applications using JavaScript.

Features of Node JS :

1. Easy Scalability: Node.js is built upon Chrome V8's engine powered by Google. It allows Node to provide a server-side runtime environment that compiles and executes JavaScript at lightning speeds.
2. Real-time web apps: Today, the web has become much more about interaction. Users want to interact with each other in real-time. Chat, gaming, constant social media updates, collaboration tools, eCommerce websites, real-time tracking apps, marketplace—each of these features requires real-time communication between users, clients, and servers across the web.
3. Fast Suite: As we have discussed that Node.js is highly scalable and lightweight, that's why it's a heavy favorite for microservice architectures. In a nutshell, microservice architectures mean breaking down the application into isolated and independent services.
4. Easy to learn and code: No matter what language you are using for the backend application, you're gonna need JavaScript for the front-end anyway. So instead of spending your time learning a server-side language such as PHP, Java, or Ruby on Rails, you can spend all your efforts learning JS and mastering it.
5. Data Streaming: Node.js comes to the rescue since it's good at handling such an I/O process, which allows users to transcode media files simultaneously while they are being uploaded. It takes less time compared to other data processing methods for processing data.
6. Corporate Support: It's an independent community aimed at facilitating the development of Node.js core tools. The foundation of Node.js was formed to speed up the development of Node.js, and it was intended to allow broad adoption of it.

Event-Driven Programming Principles:

A suite of functions for handling the events. These can be either blocking or non-blocking, depending on the implementation. Binding registered functions to events. When a registered event is received, an event loop polls for new events and calls the matching event handler(s).

characteristics:

Emitting name events: The signal that something has happened is called emitting an event. A status change in the emitting object is often the cause of this condition. Registering and unregistering listener functions: It refers to the binding and unbinding of the callback functions with their corresponding events.

Event-Driven Programming in Node.js:

- Node.js makes extensive use of events which is one of the reasons behind its speed when compared to other similar technologies. Once we start a Node.js server, it initializes the variables and functions and then listens for the occurrence of an event.
- Event-driven programming is used to synchronize the occurrence of multiple events and to make the program as simple as possible. The basic components of an Event-Driven Program are:
  - A callback function (called an event handler) is called when an event is triggered.
  - An event loop that listens for event triggers and calls the corresponding event handler for that event.

- A function that listens for the triggering of an event is said to be an 'Observer'. It gets triggered when an event occurs. Node.js provides a range of events that are already in-built. These 'events' can be accessed via the 'events' module and the EventEmitter class. Most of the in-built modules of Node.js inherit from the EventEmitter class.
- EventEmitter: The EventEmitter is a Node module that allows objects to communicate with one another. The core of Node's asynchronous event-driven architecture is EventEmitter. Many of Node's built-in modules inherit from EventEmitter.
- The idea is simple – emitter objects send out named events, which trigger listeners that have already been registered. Hence, an emitter object has two key

Analysis

While Objective-C isn't solely an imperative language, it embraces many of the imperative paradigm's characteristics, influencing its strengths, weaknesses, and features. Let's delve into this interplay:

Strengths of Imperative in Objective-C:

- Fine-grained control: You dictate every step the program takes through explicit commands and statements, allowing for precise manipulation of memory and resource utilization.
- Direct hardware interaction: Objective-C's C foundation enables low-level access to hardware, ideal for performance-critical tasks in areas like device drivers and embedded systems.
- Simple and beginner-friendly: The step-by-step nature of imperative instructions can be easier to grasp for newcomers compared to more abstract paradigms.
- Modularization through functions: Breaking down complex tasks into functions promotes code reuse and organization, improving maintainability.
- Extensive toolset: The imperative world enjoys mature libraries and frameworks, providing readily available resources for various development needs.

Weaknesses of Imperative in Objective-C:

- Error-prone: Imperative programs rely heavily on programmer decisions, increasing the risk of bugs and unexpected behavior due to manual state management.
- Complex debugging: Tracing errors can be challenging due to intricate interactions between state changes and control flow.
- Not ideal for declarative problems: For problems where the desired outcome is clear but the steps are not, imperative approaches might feel less natural to define.
- Verbose syntax: Objective-C can sometimes involve long and intricate statements compared to modern languages, potentially impacting code readability.
- Reduced focus on safety: Compared to newer languages with features like type safety and immutability, Objective-C requires more careful practices to avoid errors.

Features of Objective-C influenced by Imperative paradigm:

- Explicit state management: Variables and objects hold the program's state, manipulated through assignments and function calls.
- Control flow statements: if, while, and for loops guide program execution based on conditions and iteration.
- Procedural decomposition: Code is organized into functions for modularity and reuse.
- Sequential execution: Instructions are executed one after another in the order they are written.
- Side effects: Many operations alter the program's state outside the immediate context of the statement, requiring thoughtful consideration.

Remember, Objective-C also incorporates object-oriented concepts like classes, inheritance, and messaging, which add another layer of complexity and features.

Here's a comprehensive overview of the strengths, weaknesses, and features of event-driven programming with Node.js:

Strengths of Event-driven in Node.js:

- Efficient Resource Management: Handles multiple events concurrently without blocking, making it ideal for I/O-bound tasks and real-time applications.

- Responsiveness: Enables applications to react quickly to user interactions, data streams, or external events, enhancing user experience and responsiveness.
- Scalability: Easily accommodates increasing loads and connections by distributing events across multiple cores or servers, promoting scalability.
- Simplified Code Structure: Promotes modularity and decoupling of components as events trigger actions, improving code organization and maintainability.
- Node.js Alignment: Node.js's single-threaded, non-blocking I/O model aligns perfectly with event-driven programming, providing a natural environment for its implementation.

Weaknesses of Event-driven in Node.js:

- Complex Debugging: Asynchronous nature and interdependency of events can make debugging challenging, requiring careful tracing of event flows and potential race conditions.
- Error Handling: Handling errors effectively within an event-driven architecture requires thoughtful strategies to prevent cascading failures and unexpected behavior.
- Learning Curve: Transitioning to event-driven thinking from traditional sequential programming can involve a learning curve, especially for those unfamiliar with asynchronous concepts.

Features:

- Event Loop: The core of Node.js's event-driven architecture, continuously monitoring for events and executing corresponding callbacks.
- Asynchronous Operations: Node.js excels at handling I/O-bound tasks asynchronously, preventing blocking and maximizing resource utilization.
- Callbacks and Promises: Asynchronous operations are managed through callbacks or promises, providing mechanisms to handle results or errors when events complete.
- EventEmitter: A built-in class for creating custom events and registering listeners to respond to them, facilitating communication between components.
- Rich Ecosystem: Node.js boasts a vast ecosystem of libraries and frameworks specifically designed for event-driven programming, offering tools for various use cases.

Common use cases:

1. Real-time chat applications
2. Data streaming services
3. Web servers and APIs
4. IoT device management
5. Single-page web applications

Comparison

The Imperative and Event-Driven paradigms offer contrasting ways to navigate the choreography of your program's execution. Let's compare and contrast these approaches, illuminating their unique rhythms:

Similarities:

Both achieve outcomes: Regardless of the dance steps, both paradigms ultimately guide your program to achieve desired results. Abstraction matters: Both leverage abstraction, allowing you to focus on the bigger picture of what you want, leaving some details to the framework. Language flexibility: Many languages embrace elements of both paradigms, offering versatility and choice in your programming style.

Differences:

Control Flow:

Imperative: You orchestrate the flow, dictating each step your program takes through explicit commands and instructions. It's like painstakingly choreographing each movement in a dance routine. Event-Driven: The program reacts to events - external or internal triggers - dictating the flow of execution. It's like improvising based on cues from the music or your partner during a dance.

Focus:

Imperative: Focuses on "how" you achieve the goal, meticulously laying out the sequence of steps. Event-Driven: Focuses on "what" needs to happen in response to events, leaving the system to determine the appropriate reactions.

State Management:

Imperative: You actively manage the program's state through variables, assignments, and function calls. It's like constantly monitoring your partner's position and adjusting your own moves accordingly. Event-Driven: The state often evolves dynamically as events occur, with minimal manual manipulation. It's like relying on the rhythm and cues of the environment to guide your next step.

Examples:

Imperative: Writing a loop to check each file in a directory for a specific keyword, dictating every iteration. Event-Driven: Setting up a listener for mouse clicks on a button, defining the action to be performed in response.

Strengths:

Imperative: Fine-grained control, ideal for low-level tasks and demanding specific execution order. Event-Driven: Scalable and responsive, excels in handling concurrent events and real-time interactions.

Weaknesses:

Imperative: Can become complex and error-prone, especially for long sequences or changing requirements. Event-Driven: Debugging can be challenging, requiring tracing event chains and potential race conditions.

Challenges Faced

Diving into the world of programming paradigms online felt little difficult in the beginning. But when faced with applying that knowledge to my assignment, it satrted getting better. My given languages, Objective-C and Node.js, each brought their own unique paradigm puzzle. Initially had few confusions when i came into notice that Objective c doesn;t follow imperative paradigm as mentined but then i got cleared with that doubt.

However, I soon discovered that mastering paradigms, like cooking or any other skill, requires a slow and deliberate approach. Instead of attempting grand comparisons, I focused on deconstructing each language's paradigm piece by piece. I read practical guides, went through clear tutorial websites.

Node.js related information and notes were availbale over internet but very less about objective c. So had to depend on Chat AI bots for more understanding of concepts.

Conclusion

In conclusion, this exploration of programming paradigms through the lenses of Objective-C and Node.js revealed a fascinating landscape of diverse approaches to code. While the imperative nature of Objective-C offers fine-grained control and a familiar step-by-step structure, Node.js's embrace of the event-driven paradigm shines in its responsiveness and scalability. This journey highlighted the value of understanding various paradigms, equipping us to choose the right tool for the job. . With this newfound knowledge, we embark on the future of programming, ready to craft elegant and efficient solutions for any challenge that may arise.

References

- https://www.w3schools.com/nodejs/nodejs_intro.asp
- https://www.geeksforgeeks.org/nodejs/
- https://www.geeksforgeeks.org/difference-between-c-and-objective-c/
- https://www.geeksforgeeks.org/explain-event-driven-programming-in-node-js/
- https://www.tutorialspoint.com/objective_c/index.htm
- https://www.devmaking.com/learn/programming-paradigms/imperative-paradigm/
- https://www.geeksforgeeks.org/introduction-of-programming-paradigms/
- https://www.studysmarter.co.uk/explanations/computer-science/computer-programming/event-driven-programming/