PARADIGMAS DE PROGRAMACIÓN

PROGRAMMING PARADIGMS

Autor 1: Aldair Estiven Lasso Acosta

Autor 2: Diego Alejandro Castro Cardona

*Ingeniería de Sistemas, Universidad Tecnológica de Pereira*

Correo-e: [d.castro4@utp.edu.co](mailto:d.castro4@utp.edu.co)

Correo-e: [aldair.lasso@utp.edu.co](mailto:aldair.lasso@utp.edu.co)

***Resumen*— Breve descripción de loa paradigmas que existen o que se encuentran presentes al día de hoy en su completo funcionamiento alrededor del mundo y sus distintas capacidades y cualidades. Pasando por la mayoría de los paradigmas notando que cada uno de ellos se basa en una manera distinta una de la otras pasa el desarrollo de una misma tarea e incluso la facilidad que puede brindar cierto paradigma al darle el uso en el momento debido o situación a la cual se ajusta cada uno de estos para hacer de una manera más fácil el trabajo de los desarrolladores y productores de la tecnología.**

**Un paradigma de programación es un estilo de desarrollo de programas. Es decir, un modelo para resolver problemas computacionales. Los lenguajes de programación, necesariamente, se encuadran en uno o varios paradigmas a la vez a partir del tipo de órdenes que permiten implementar, algo que tiene una relación directa con su sintaxis.**

**¿Cuáles son los principales paradigmas de programación?**

* **Imperativo. Los programas se componen de un conjunto de sentencias que cambian su estado. Son secuencias de comandos que ordenan acciones a la computadora.**
* **Declarativo. Opuesto al imperativo. Los programas describen los resultados esperados sin listar explícitamente los pasos a llevar a cabo para alcanzarlos.**
* **Lógico. El problema se modela con enunciados de lógica de primer orden.**
* **Funcional. Los programas se componen de funciones, es decir, implementaciones de comportamiento que reciben un conjunto de datos de entrada y devuelven un valor de salida.**
* **Orientado a objetos. El comportamiento del programa es llevado a cabo por objetos, entidades que representan elementos del problema a resolver y tienen atributos y comportamiento.**

***Palabras clave—* Tecnología, Informática, Tics, Computación, paradigmas, solución de problemas, desarrollo, paradigmas de programación, Lenguajes de programación.**

***Abstract*— A brief description of the paradigms that exist or that are present today in their complete functioning around the world and their different capacities and characteristics. Going through most of the paradigms noting that each of them is based in a different way one of the others passes the development of the same task and even the ease that a certain paradigm can provide by giving it the use at the right time or situation. which adjusts each of these to make the work of developers and producers of technology easier.**

**A programming paradigm is a style of program development. That is, a model to solve computational problems. The programming languages, necessarily, are framed in one or several paradigms at the same time based on the type of orders they allow to implement, something that has a direct relationship with their syntax.**

**What are the main programming paradigms?**

* **Imperative. Programs are made up of a set of sentences that change their status. They are scripts that order actions to the computer.**
* **Declarative. Opposite the imperative. The programs describe the expected results without explicitly listing the steps to be taken to achieve them.**
* **Logical. The problem is modeled with first-order logic statements.**
* **Functional. Programs are made up of functions, that is, behavioral implementations that receive a set of input data and return an output value.**
* **Object oriented. The behavior of the program is carried out by objects, entities that represent elements of the problem to be solved and have attributes and behavior.**

***Key Word* —Technology, Computing, paradigms, problem solving, development, programming paradigms, programming languages.**

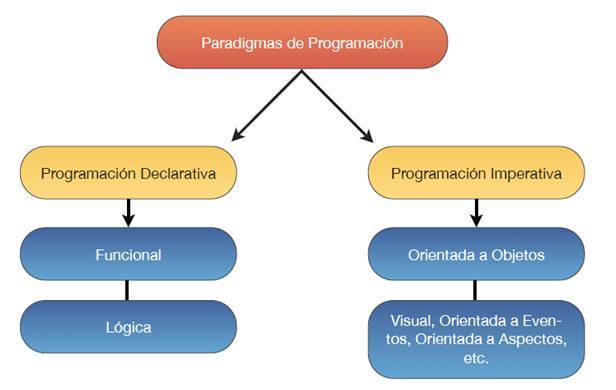
1. INTRODUCCIÓN

En el presente documento vamos a mostrar una breve descripción acerca de los paradigmas de programación, pero tratando de enfatizar sobre los conceptos y estructuras básicas de los paradigmas de programación, lenguajes de programación, algunos tipos de paradigmas de programación en qué consisten, características.

A continuación, también vamos a enumerar una cierta cantidad de paradigmas de programación, sus definiciones, ¿cómo se usan?, teorías simples, ventajas, etc. Adicionalmente agregaremos algunos lenguajes que pertenecen a los tipos de paradigmas y unos ejemplos de los mismos para tratar de comprender su funcionalidad y cómo éstos influyen en el desarrollo del software.

1. CONTENIDO

A lo largo de la historia, el término «paradigma» fue objeto de muchas interpretaciones. En su origen griego, significaba «modelo», «ejemplo» o «patrón». Sobre este punto de partida, podemos hablar de un paradigma como un conjunto de creencias, prácticas y conocimientos que guían el desarrollo de una disciplina durante un período de tiempo. En diversas ramas de la ciencia, un conjunto de ideas en vigencia puede ser reemplazado drásticamente por otro que entre en conflicto con él y se demuestre más acertado. La programación tiene sus propios paradigmas, pero el término «paradigma de programación» no necesariamente representa un modelo único que deba ser respetado hasta que aparezca otro mejor.



**Programación Imperativa:**

La programación imperativa es la forma natural de programar un computador, es el estilo de programación que se utiliza en el ensamblador, el estilo más cercano a la arquitectura del computador.

Se basa en dar instrucciones al ordenador de cómo hacer las cosas en forma de algoritmos, Es un paradigma de programación en términos del estado del programa y sentencias que cambian dicho estado y se le debe explicar a la computadora de manera detallada las instrucciones de cómo realizar una tarea, en otras palabras, la programación imperativa no va realizar tareas que no estén declaradas.

La programación imperativa es la más usada y la más antigua, el ejemplo principal es el lenguaje de máquina. Ejemplos de lenguajes puros de este paradigma serían el C, BASIC o Pascal.

**¿Como se usa?**

La implementación de hardware de la mayoría de computadores es imperativa; prácticamente todo el hardware de los computadores está diseñado para ejecutar código de máquina, que es nativo al computador, escrito en una forma imperativa. Esto se debe a que el hardware de los computadores implementa el paradigma de las Máquinas de Turing. Desde esta perspectiva de bajo nivel, el estilo del programa está definido por los contenidos de la memoria, y las sentencias son instrucciones en el lenguaje de máquina nativo del computador (por ejemplo, el lenguaje ensamblador).

Los lenguajes imperativos de alto nivel usan variables y sentencias más complejas, pero aún siguen el mismo paradigma. Las recetas y las listas de revisión de procesos, a pesar de no ser programas de computadora, son también conceptos familiares similares en estilo a la programación imperativa; cada paso es una instrucción, y el mundo físico guarda el estado (Zoom).

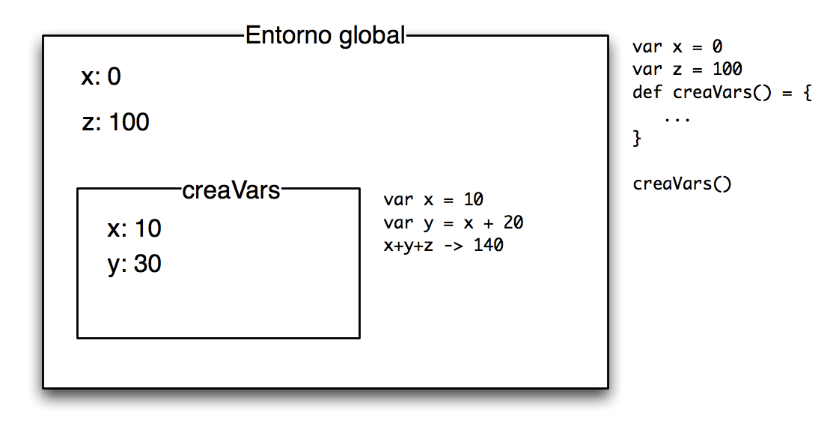
**Características de la arquitectura arquitectura clásica de Von Newmann:**

* memoria donde se almacenan los datos (referenciables por su dirección de memoria) y el programa
* unidad de control que ejecuta las instrucciones del programa (contador del programa)
* Los primeros lenguajes de programación (como el Fortran) son abstracciones del ensamblador y de esta arquitectura, lenguajes más modernos como el BASIC o el C han continuado esta idea

**Algunos lenguajes imperativos.**

* ASP
* BASIC
* C
* Fortran
* Pascal
* Perl
* PHP
* Lua
* Java

**Ejemplos:**



function fibonacci(n) {

var actual, ant1, ant2;

if (n === 0) {

actual = 0;

} else if (n === 1) {

actual = 1;

} else {

ant1 = ant2 = 1;

for (i = 2; i < n; i++) {

actual = ant1 + ant2;

ant2 = ant1;

ant1 = actual;

}

}

return actual;

}

-module(fibonacci).

-export([fibonacci/1]).

-spec fibonacci(non\_neg\_integer()) -> non\_neg\_integer().

fibonacci(0) -> 0;

fibonacci(1) -> 1;

fibonacci(N) -> fibonacci(N - 1) + fibonacci(N - 2).

**Programación Declarativa:**

en contraposición a la programación imperativa.

Es un paradigma de programación basado en el desarrollo de programas “declarando” un conjunto de condiciones, proposiciones, afirmaciones, restricciones, ecuaciones o transformaciones que describen un problema, pero no las instrucciones necesarias para resolverlos. La solución se obtiene mediante mecanismos internos, este tipo de programación resuelve problemas si la necesidad de escribir las declaraciones al pie de la letra.

La solución es obtenida mediante mecanismos internos de control, sin especificar exactamente cómo encontrarla (tan sólo se le indica a la computadora que es lo que se desea obtener o que es lo que se está buscando). No existen asignaciones destructivas, y las variables son utilizadas con Transparencia referencial

**Diferencia entre la programación imperativa y declarativa:**

En la programación imperativa se describe paso a paso un conjunto de instrucciones que deben ejecutarse para variar el estado del programa y hallar la solución, es decir, un algoritmo en el que se describen los pasos necesarios para solucionar el problema.

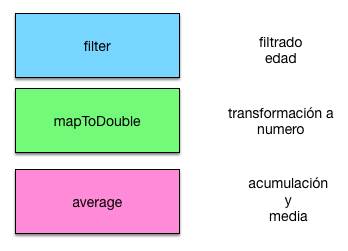
En la programación declarativa las sentencias que se utilizan lo que hacen es describir el problema que se quiere solucionar, pero no las instrucciones necesarias para solucionarlo. Esto último se realizará mediante mecanismos internos de inferencia de información a partir de la descripción realizada.

**Ventajas**

Se ha dicho que los lenguajes declarativos tienen la ventaja de ser razonados matemáticamente, lo que permite el uso de mecanismos matemáticos para optimizar el rendimiento de los programas. Son fiables, elegantes y expresivos.

**Algunos lenguajes declarativos:**

* Haskell (Programación funcional)
* ML (Programación funcional)
* Lisp (Programación funcional)
* Prolog (Programación Lógica)
* F-Prolog (Programación Lógica Difusa)
* Curry (Programación Lógico-Funcional)



package com.arquitecturajava;

import java.util.ArrayList;

import java.util.List;

import java.util.Optional;

import java.util.OptionalDouble;

public class Principal2 {

public static void main(String[] args) {

Persona p1= new Persona("pepe",20);

Persona p2= new Persona("juan",12);

Persona p3= new Persona("angela",30);

Persona p4= new Persona("gema",40);

Persona p5= new Persona("david",15);

List<Persona> lista=new ArrayList<Persona>();

lista.add(p1);

lista.add(p2);

lista.add(p3);

lista.add(p4);

lista.add(p5);

OptionalDouble resultado=lista.stream().filter(persona->persona.getEdad()>=18)

.mapToInt(persona->persona.getEdad())

.average();

if(resultado.isPresent()) {

System.out.println(resultado.getAsDouble());

}

}

}

**Programación Orientada o Objetos:**

es un enfoque de programación que combina datos y acciones asociadas (métodos) en estructuras lógicas (objetos). Este enfoque aumenta la capacidad para administrar la complejidad del software, lo cual resulta especialmente importante cuando se desarrollan y mantienen aplicaciones y estructuras de datos de gran tamaño.

Es un paradigma de programación que usa objetos en sus acciones para diseñar aplicaciones y programas informáticos.

Está basada en varias técnicas, incluyendo herencia, cohesión, abstracción, polimorfismo, acoplamiento y encapsulamiento. Su uso se popularizó a principios de la década de 1990. En la actualidad, existe una gran variedad de lenguajes de programación que soportan la orientación a objetos.

Es una forma especial de programar, este paradigma de programación es cercano a como expresamos las cosas en la vida real en nuestro día a día. Con la POO tenemos que aprender a pensar las cosas de una manera distinta para poder escribir nuestros programas en términos de objetos, propiedades y métodos.

**¿Qué es un objeto?**

Los objetos son como variables, pero con más valores.

Pensar en términos de objetos es pensar de manera natural como lo haríamos en la vida real. Por ejemplo, al pensar en un coche para modelarlo en un esquema de POO. Diríamos que el coche es el elemento principal que tiene una serie de características, como podrían ser el color, el modelo o la marca. Además, tiene una serie de funcionalidades o métodos asociados al mismo, como pueden ser ponerse en marcha, parar o acelerar.

En un esquema POO el coche sería el objeto, las propiedades serían las características como el color o el modelo y los métodos serían las funcionalidades asociadas como ponerse en marcha o parar.

**Algunos lenguajes orientados a objetos:**

Ada

C++

C#

VB.NET

Clarion

Delphi

Eiffel

Java

Lexico (en castellano)

Objective-C

Ocaml,

Oz

PHP

PowerBuilder

Python

Ruby

Smalltalk.



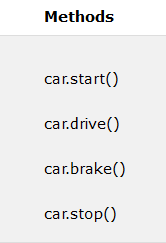
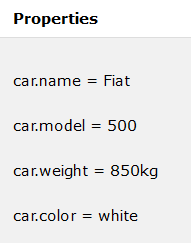


Imagen: https://www.w3schools.com/js/js\_objects.asp

**Ventajas:**

* Los componentes se pueden reutilizar.
* Facilidad de mantenimiento y modificación de los objetos existentes.
* Una estructura modular clara se puede obtener, la cual no revelará el mecanismo detrás del diseño.
* Se proporciona un buen marco que facilita la creación de rica interfaz gráfica de usuario aplicaciones (GUI).
* Se acopla bien a la utilización de bases de datos, debido a la correspondencia entre las estructuras.

La programación orienta a objetos es una forma especial de pensar, nos permite modelar nuestra realidad dentro de un entorno de programación, partiendo de la primicia de que todo en esta vida es un objeto programable.

**Paradigma funcional:**

Es una manera de las cuales podemos resolver una gran variedad de problemas, en los momentos en los cuales se esté desarrollando software utilizando este paradigma, se estará empleando principalmente con funciones, se evitará el uso de datos mutables, así como el hecho de compartir estados entre funciones.

La programación funcional es un paradigma declarativo. Pues nos enfocaremos en qué estamos haciendo y no en cómo se está haciendo que sería el enfoque imperativo. Esto quiere decir que nosotros expresaremos nuestra lógica sin describir controles de flujo; no usaremos ciclos o condicionales.

Al trabajar con programación funcional, nos toparemos con conceptos tales como:

Mutabilidad.

Efecto secundario.

Funciones puras.

Composición de funciones.

Estados compartidos.

Citare brevemente aspectos acerca de los puntos anteriormente tratados:

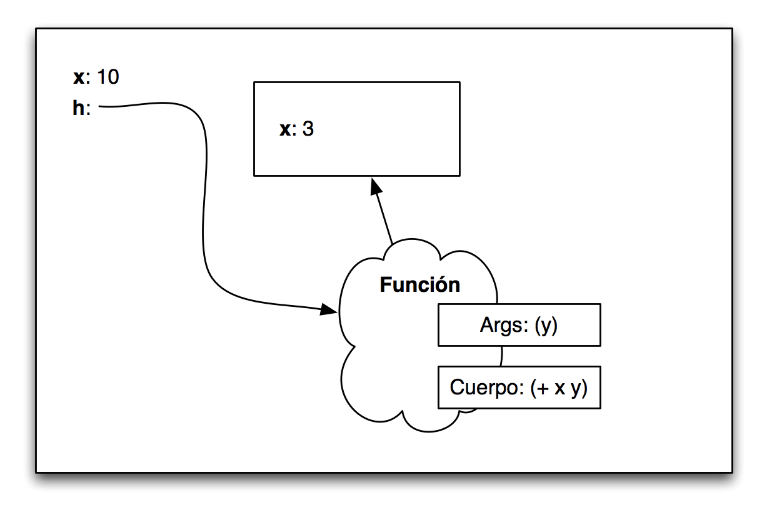
Mutabilidad; esto hace referencia a un objeto el cual no puede ser modificado una vez se haya creado. Un ejemplo que se podría citar seria, una variable en java, pues este no es n objeto mutable.

Efectos secundarios; Un efecto secundario seria cualquier cambio de estado en la aplicación que sea observable fuera de la función llamada. Los efectos secundarios se evitan principalmente en la programación funcional, para tener como resultado programas mucho más fáciles de comprender y fáciles de probar.

Funciones puras; son funciones las cuales dando el mismo input siempre retornan el mismo output, estas no tienen efectos secundarios.

La composición de funciones se denomina como el proceso de combinar dos o más funciones, para ejecutar cada una de estas funciones en secuencia para obtener un resultado en concreto.

El estado compartido es cualquier variable, objeto o espacio de memoria que exista en un ámbito compartido. A menudo, en la programación orientada a objetos, los objetos se comparten entre ámbitos al agregar propiedades a otros objetos.



Actualmente contamos con una gran cantidad de lenguajes en donde podemos hacer uso de la programación funcional, por ejemplo:

Java

PHP

Ruby

Python

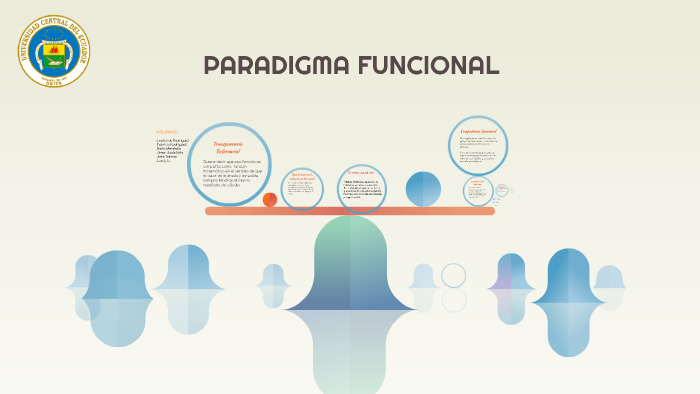
Elixir

Kotling

Haskell

Erlang

Muchos de estos lenguajes no están cien por ciento enfocados en la programación funcional, lo cual no es malo, de hecho, estos nos permiten hacer una transición de un paradigma a otro, comúnmente de la programación orientada a objetos ha programación funcional.



**Programación lógica:**

La programación lógica estudia el uso de la lógica para el planteamiento de problemas para alcanzar la solución automática.

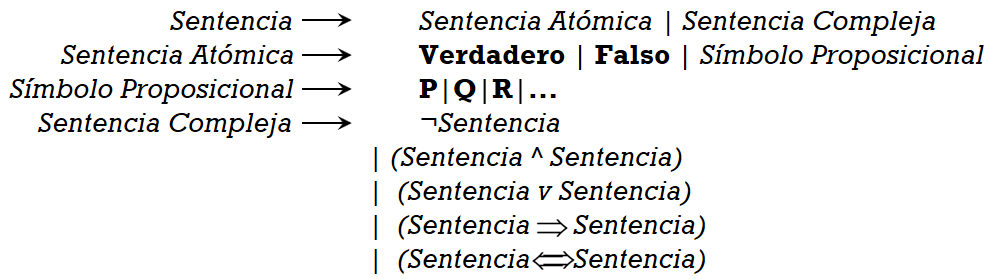
Que cosas trata de resolver este tipo de programación; Dado un problema de cualquier tipo, saber si la afirmación, por ejemplo, llamemos a la información b es solución o no del problema o en qué casos lo es. Además, haciendo que los métodos sean implantados en máquinas de manera que la resolución del problema se haga de forma automática

Entonces diría que la programación lógica construye base de conocimientos mediante reglas y hechos.

Algunas de las características de este tipo de paradigmas son; la unificación de términos, mecanismos de inferencia automática, Recursión como estructura de control básica.

Visión lógica de la computación.

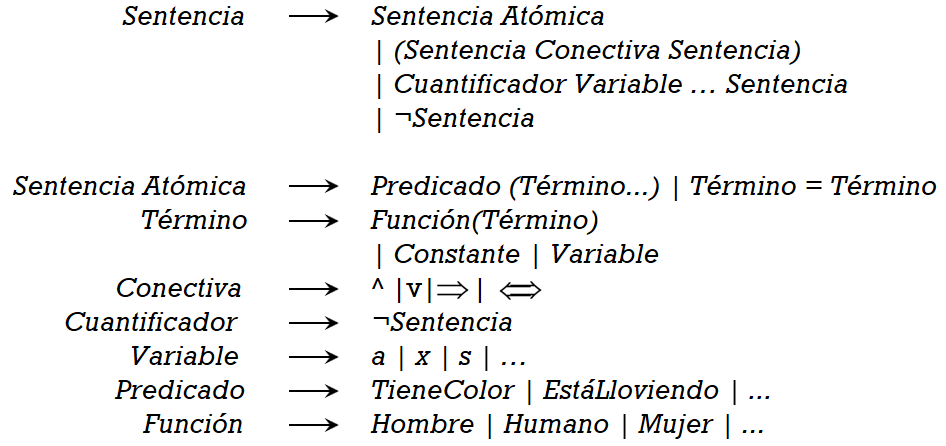
Lógica Proposicional



La lógica proposicional trata con sistemas lógicos que carecen de cuantificadores, o variables interpretables como entidades. En lógica proposicional si bien no hay signos para variables de tipo entidad, sí existen signos para variables proposicionales (es decir, que pueden ser interpretadas como proposiciones con un valor de verdad definido), de ahí el nombre proposicional. La lógica proposicional incluye además de variables interpretables como proposiciones simples signos para conectivas lógicas, por lo que dentro de este tipo de lógica puede analizarse la inferencia lógica de proposiciones a partir de proposiciones, pero sin tener en cuenta la estructura interna de las proposiciones más simples.

Lógica de primer orden

Este se basa en lenguaje lógico matemático formal, su estructura seria de la siguiente manera;



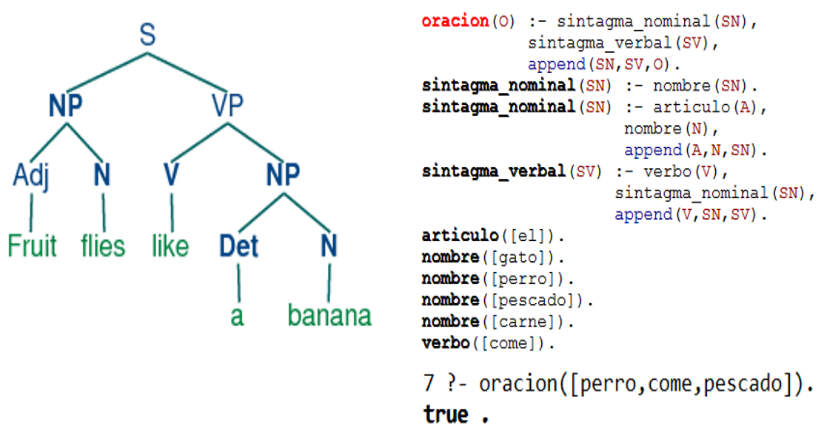
Es un sistema de la manera formal, empleado àra estudiar inferencias de prmer orden. La lógica de primer orden tiene el poder expresivo suficiente para definir a prácticamente todas las matemáticas.

Conceptos claves de el paradigma denominado paradigma lógico; hechos: Sobre las expresiones se puede definir el concepto de aridad como la cantidad de objetos presentes en su declaración, a partir de este concepto se pueden clasificar los hechos como:

Enunciado: Aridad = 0. Ejm: Verdadero.

Propiead: Aridad = 1. Ejm: humano(Pepito), animal(Perro).

Relación: Aridad > 1. Ejm: padre(Homero, Bart), aprobar(Juan, Lenguajes de Programación).



**Programación reactiva:**

Este tipo de programación en la actualidad proporciona tiempos de respuesta rápidos y de una manera constante, definido limites en el tiempo de las respuestas, de manera que los problemas pueden ser detectados de manera rápida y eficaz. Este sistema permanece responsivo incluso en situaciones de error, y ante incrementos en la carga de trabajo. Se basa en el intercambio de mensajes asíncronos.

Con el tiempo este tipo de programación se ha ido ganando un lugar en la comunidad de desarrolladores y de los clientes gracias a su oferta de características para la construcción de aplicaciones de manera declarativa las mismas que hacen que nuestras aplicaciones sean más sensibles y resilintes. Se integran los posibles fallos en el modelo de programación, de forma que, al estar dentro del diseño desde el inicio, es posible reaccionar a ellos y solucionarlos. Así, las aplicaciones reactivas son altamente tolerantes a los fallos (son responsivas frente a ellos) y pueden repararse a ellas mismas en el tiempo de ejecución.

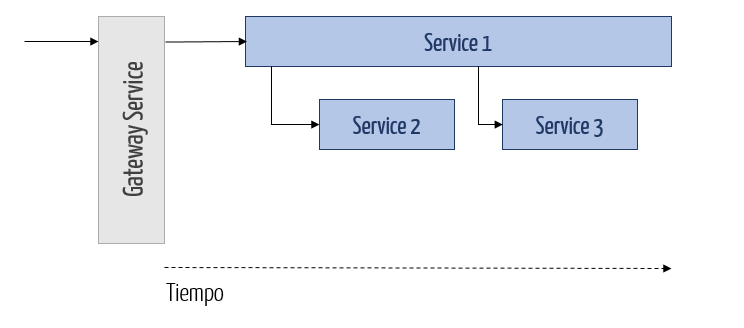
Puede expandirse de acuerdo con su utilización, gracias a la elasticidad (opción de añadir, o remover nodos según petición) que se le añade. Gracias a la escalabilidad, se pueden minimizar los costes de tener aplicaciones en la nube y gestionar los riesgos (tener pocos recursos lleva a perder clientes, tener demasiados puede resultar costoso). Además, así los Sistemas Reactivos pueden reaccionar a variaciones en la carga de trabajo y a cambios en la frecuencia de peticiones incrementando o reduciendo los recursos asignados para servir dichas peticiones. Para ello, hay que diseñar una arquitectura sin puntos de contención o cuellos de botella centralizados, resultando en la capacidad de dividir o replicar componentes y distribuir las peticiones entre ellos. Los Sistemas Reactivos soportan algoritmos de escalado predictivos y Reactivos, al proporcionar relevantes medidas de rendimiento en tiempo real.

Los Sistemas Reactivos confían en este intercambio de mensajes asíncrono para establecer fronteras entre componentes. Esto asegura bajo acoplamiento, aislamiento y transparencia de ubicación. Además, estas fronteras permiten delegar fallos como mensajes.

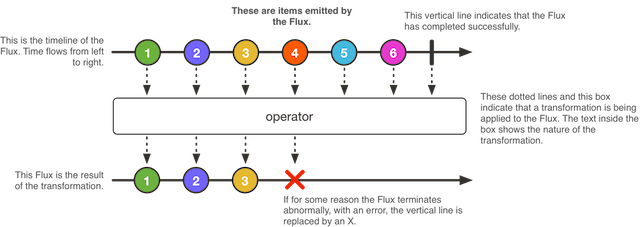
El intercambio de mensajes (realizado mediante el modelado y monitorización de las colas de mensajes y la aplicación de back-pressure) permite gestionar la carga, la elasticidad y el control de flujo. Además, al utilizar la mensajería basada en ubicaciones transparentes, podemos gestionar fallos y trabajar con los mismos bloques y semánticas a través de un clúster o dentro de un solo nodo.

Conclusiones

Para crear un sistema reactivo, debemos tener en cuenta que integrará estas cuatro características interrelacionadas, es decir, que será responsiva, resiliente escalable y orientada a mensajes.



En el concepto de programación reactiva se manejan flujos de datos asíncronos entre productores de datos y consumidores de datos en donde ambos necesitan reaccionar a esos datos de forma que no bloqueen los recursos. Entonces la característica principal de la programación reactiva es que no bloquea ningún recurso y se maneja a través de eventos asíncronos que requieren de pocos hilos para escalar.



1. CONCLUSIONES

En conclusión, los paradigmas de programación son diferentes tipos de herramientas que nos permiten desarrollar software, la comprensión básica de los conceptos de los lenguajes de programación y los diferentes paradigmas son necesarios para un lenguaje de programación, ya que cada paradigma pretende una cierta lógica lo cual en algunos casos un software va más de un tipo de paradigma.

Los lenguajes de programación influencian notablemente la manera en que pensamos acerca del diseño y construcción del software y los algoritmos y estructuras de datos que utilicemos para desarrollar software.

REFERENCIAS

1. <https://esacademic.com/dic.nsf/eswiki/715531>
2. <https://prezi.com/7vcuauwjiqzf/programacion-declarativa-vs-programacion-imperativa/>
3. <http://www.dccia.ua.es/dccia/inf/asignaturas/LPP/2010-2011/clases-domingo/sesion20.html>
4. <https://medium.com/@Loopa/paradigmas-de-programaci%C3%B3n-programaci%C3%B3n-imperativa-y-programaci%C3%B3n-declarativa-4c4a4182fd87>
5. <https://www.ecured.cu/Programaci%C3%B3n_Imperativa>
6. <https://aprendiendoarduino.wordpress.com/tag/programacion-imperativa/>
7. <https://esacademic.com/dic.nsf/eswiki/899529>
8. <http://www.larevistainformatica.com/lenguajes-programacion-orientada-objetos.html>
9. <https://codigofacilito.com/articulos/programacion-funcional>
10. <https://profile.es/blog/que-es-la-programacion-reactiva-una-introduccion/>
11. <https://www.4rsoluciones.com/blog/que-son-los-paradigmas-de-programacion-2/>