PARADIGMAS DE PROGRAMACIÓN

Un paradigma de programación es un caso específico de paradigma. La definición estricta de paradigma se puede encontrar en la bibliografía pero intuitivamente nos podemos quedar con la idea de que **un paradigma es un ejemplo** clave que extrae las características definitorias de algo. Por ejemplo, si decimos que Espronceda es el paradigma de los autores románticos queremos decir que las características que destacan en este autor son las características que definen el ser de un autor romántico. Por tanto, al decir que otro autor es romántico basta con pensar en las características de Espronceda como paradigma y ya sabremos cómo es este nuevo autor.

Por esto decimos que un paradigma es un ejemplo base sobre el que considerar los demás individuos. También podemos decir que es un ejemplo de referencia al que se acude para valorar si una cierta situación cumple o no con las características esperadas. Por tanto, es una clasificación, separando los individuos que cumplen las condiciones y los que no.

Esta aproximación nos da una idea del concepto pero no es suficiente. Desde un punto de vista de ingeniería podemos contar con la siguiente definición:

> "Un paradigma es un conjunto de teorías, estándares y métodos que juntos representan una forma de organizar el conocimiento, esto es, una forma de ver el mundo" [Thomas Kuhn]

Esta definición puede parecer larga, pero realmente es muy sintética ya que cada una de sus palabras encierra un significado complejo. Empezaremos por simplificarla en:

> "es un conjunto de elementos que representan una forma de organizar el conocimiento".

Con esto estamos diciendo que el paradigma nos va a ayudar a organizar un conocimiento sobre algo. También nos dice que esta organización representará una forma de ver el mundo. Esto quiere decir que **un paradigma es una abstracción** de una cierta realidad. Sabemos que toda abstracción y toda organización implican una simplificación pero también que nos ayudan a tratar situaciones cuya complejidad sería inabordable de otro modo.

Siguiendo con el análisis nos queda tratar los **tres elementos** que componen un paradigma: **teorías**, **estándares** y **métodos**.

Estos elementos se interrelacionan en el paradigma para constituirlo de la siguiente forma: las teorías representan los conceptos que van a formar la abstracción, los estándares establecen normas que permitan realizar la abstracción según unas definiciones que las hagan compatibles con el resto de los trabajos existentes y los métodos nos dan un conjunto de pasos a dar para poder aplicar las teorías y estándares sin necesidad de hacer uso de improvisaciones.

Como vemos, esta definición nos dice que un paradigma es una herramienta de gran utilidad en las aplicaciones de ingeniería como ayuda para llevar a buen término conjuntos de tareas similares con las características definidas por el paradigma.

Pero, un Paradigma de Programación es algo más específico y se usa para cualificar conceptos muy dispares dentro del área de aplicación de los lenguajes.

Sin entrar en valorar la corrección en el uso de la denominación de paradigma de programación que se considera incorrecto por algunos autores, profundizaremos en su significado a través de las distintas aplicaciones o usos que del concepto se hacen. Veremos cómo de este estudio se obtienen jugosas conclusiones que enfocan perfectamente el ámbito de estudio sobre el que queremos trabajar al plantear como objetivo los paradigmas funcional, paralelo y de tiempo real.

Como primera idea, vamos a decir que **un paradigma de programación es una forma de pensar a la hora de programar**, una manera de construir programas. A lo largo de la historia, algunos lenguajes de programación han introducido nuevos paradigmas de programación, como pasó con Fortran (1957 - imperativo), Lisp (1959 - funcional), Prolog (1972 - lógico) o Simula (1979 - orientado a objetos), y otros muchos han cumplido con los paradigmas existentes, con uno o varios y con distintos niveles de compromiso.

Con esta primera idea de Paradigma de Programación (en adelante **PdP** para mayor claridad del texto) vamos a colocar los paradigmas en su lugar dentro de un ciclo de desarrollo de software (principal actividad donde se incluyen tareas de programación y, por tanto, se usan PdP). Vamos a tomar el siguiente diagrama como esquema simplificado de un desarrollo software.

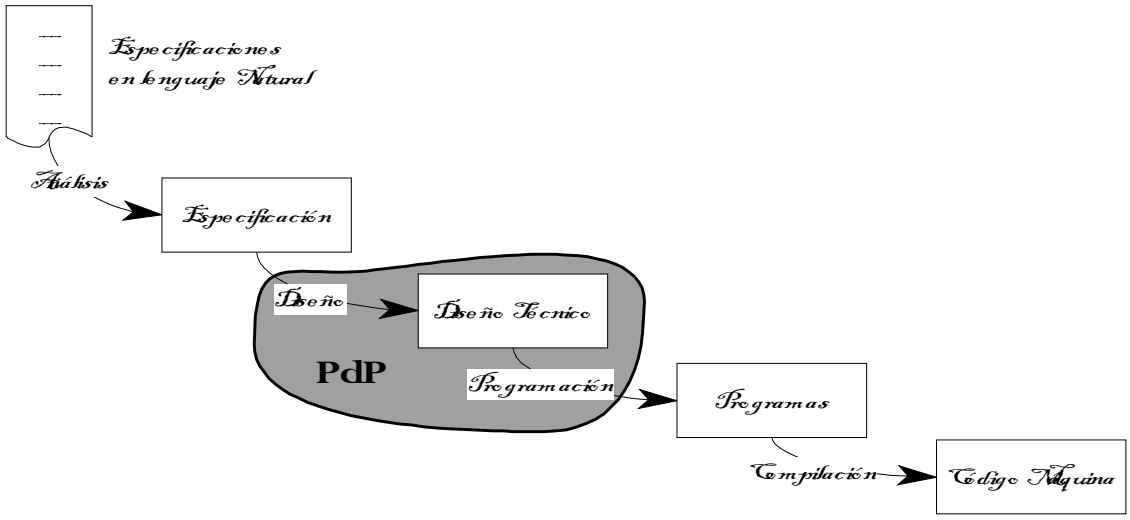

Figura 2.1: PdP en un ciclo de desarrollo software

Como se puede apreciar en el diagrama partimos de unas especificaciones que estarán en lenguaje natural. No hay que confundir esta denominación con lenguaje humano hablado o escrito, realmente lo que significa, en este caso, es algún tipo de dialecto distinto de cualquiera que el ordenador pueda entender directamente. Un ejemplo de lenguaje natural en este sentido puede ser el lenguaje matemático, manejado por humanos pero inútil para un ordenador (de forma directa, salvo que se use un programa matemático especial). También podemos considerar lenguaje natural una lista de especificaciones en un documento, un libro, una imagen, un diagrama o una presentación. Este lenguaje natural se caracteriza por su fácil comprensión para el ser humano, pero también por su imprecisión y ambigüedad.

Los siguientes pasos existentes son el análisis y el diseño en los que se utilizan técnicas especiales de ingeniería (del software, del conocimiento u otras) que permitan abordar el proyecto con calidad, planificando y realizando las tareas en orden.

Como tercer paso tenemos la programación, donde son de utilidad los PdP. En este paso se genera código en un lenguaje de programación no comprensible para el ordenador pero directamente traducible (salvo errores sintácticos).

El último paso es la generación de código ejecutable por el ordenador, comúnmente denominado código máquina.

Dentro de este ciclo **el paradigma sirve como guía a la hora de implementar en un cierto lenguaje** de programación lo diseñado como respuesta a los requerimientos recibidos y también para elegir un lenguaje de implantación según las necesidades que el diseño establece.

Se pueden considerar los siguientes PdP y algún otro, ya que esta enumeración no es exhaustiva:

- Programación concurrente.
- Programación declarativa.
- Programación distribuida.
- Programación de tiempo real.
- Programación estructurada.
- Programación funcional.
- Programación imperativa.
- Programación lógica.
- Programación modular.
- Programación orientada a módulos.
- Programación orientada a objetos.
- Programación orientada a procedimientos.
- Programación cliente/servidor.

En este punto ya tenemos alguna definición y hemos aclarado el concepto de PdP. Tenemos incluso ejemplos que nos relacionan el concepto con otros conocimientos que han sido adquiridos en otros momentos del aprendizaje de la programación. Pero, aun queda pendiente la utilidad concreta de los PdP. **¿Para qué nos sirven los paradigmas?** A esta pregunta intentaremos dar respuesta en el siguiente apartado.

Utilidad y cumplimiento de los PdP

La utilidad de los paradigmas desde un punto de vista práctico es triple:

- Sirven para **establecer clasificaciones** dentro del conjunto de los lenguajes de programación. Esta organización es de utilidad como ayuda para comprender la evolución, relaciones y características de los elementos clasificados.
- Ayudan a **describir lenguajes** existentes en base a los paradigmas que cumplen lo que facilita tareas de ingeniería del software como la elección del lenguaje de implementación. Gracias a una elección basada en los paradigmas cumplidos por los lenguajes candidatos, se podrá garantizar que el lenguaje permita realizar las tareas requeridas.
- Permiten establecer las **características deseadas de un nuevo lenguaje** en función de los

paradigmas que deseamos que cumpla. Gracias a lo cual se puede planificar las características para que se ajuste a lo deseado y a lo requerido por los futuros usuarios del lenguaje. Se podrán crear lenguajes que cubran segmentos no soportados por otros cuando sea necesario y el estudio de los paradigmas necesarios brindará la oportunidad de conocer a priori como será el lenguaje antes de realizar el esfuerzo de su creación.

Para conseguir esta utilidad por parte de los paradigmas, hay que establecer también el concepto de cumplimiento de un paradigma. Los lenguajes de programación cumplirán o no un paradigma según cumplan con las características esperadas.

Pero ¿qué sucede cuando un lenguaje no cumple todas pero sí parte de las características? Para enfrentar estas situaciones necesitamos relajar el concepto de cumplimiento o introducir variantes del mismo. Definiremos los **niveles de cumplimiento** de un paradigma como los siguientes:

- **Paradigma no permitido**. En este caso, un determinado paradigma no puede ser utilizado o conseguido con este lenguaje porque las construcciones del mismo no lo permiten.
- **Paradigma permitido**. A este nivel, es posible programar siguiendo cierto paradigma pero no es sencillo o cómodo y en algunos casos es especialmente difícil porque el lenguaje no favorece su utilización.
- **Paradigma soportado**. Diremos que está soportado cuando es la recomendación más común para el lenguaje, cuando la programación habitual lo sigue e incluso resulta complejo y difícil no cumplir el paradigma.
- **Paradigma obligatorio**. En este caso será imposible no seguir el paradigma porque no hay construcciones en el lenguaje que permitan salir de la norma que define el paradigma.

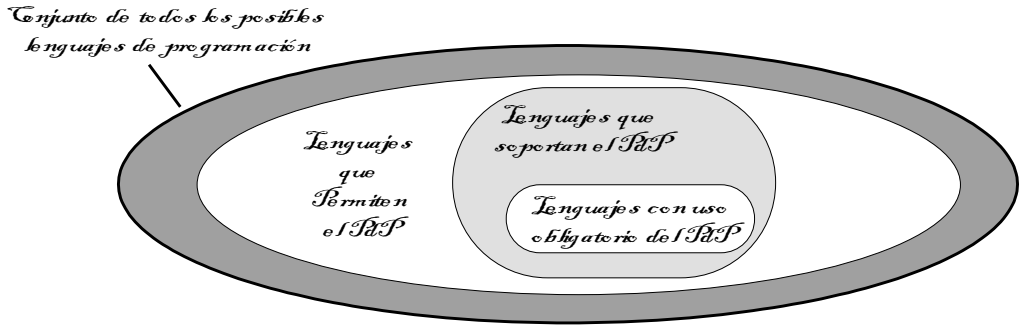

Figura 2.2: Grado de cumplimiento de un PdP.

Para ilustrar los niveles de cumplimiento de un paradigma analizaremos cómo cumplen ciertos lenguajes los paradigmas con los que se les relaciona.

- Pascal, y la programación estructurada. Este lenguaje, soporta la programación estructurada ya que es la forma preferida de programar para el lenguaje, pero es posible utilizar la orden "*goto*" y romper con este tipo de programación. Casi es imposible encontrar ejemplos de programas en Pascal que usen esta orden, salvo el caso de los que sirven para ilustrar su sintaxis.
- Java, y la programación orientada a objetos. Este paradigma es obligatorio en Java, no es posible realizar ni un solo programa que no defina una clase y use objetos.

- C, y la programación estructurada. Que está permitida pero no soportada. La orden "*return*" y la orden "*break*" permiten establecer varios puntos de salida en los bloques de código y funciones lo que rompe la programación estructurada. El uso de estas órdenes está muy extendido y es casi necesario por la forma como está creado el lenguaje.
- Fortran, y la programación orientada a objetos. En este caso, se trata de un paradigma no permitido ya que no existe ninguna forma de crear objetos, clases u otros elementos de la programación orientada a objetos.

	Obligado	Soportado	Permitido	Prohibido
Paradigma de programación estructurada	Caml	Pascal	java C	Fortran
Paradigma de programación orientado a objetos	Java	C++	Pascal	Fortran

Figura 2.3: Niveles de cumplimiento de paradigmas de algunos lenguajes

PERSPECTIVAS SOBRE PDP

Los paradigmas establecen un nivel superior de trabajo sobre los lenguajes de programación, pero también podemos considerar una perspectiva aún más elevada en la que creamos organizaciones o estudiamos las relaciones existentes entre los PdP.

Estas relaciones se pueden observar desde distintas perspectivas según se atienda a los distintos matices de cada paradigma. Veremos que hay varias posibilidades, que las relaciones son complejas e incluso que son algo difíciles de establecer en sus últimas consecuencias.

De entre las posibles perspectivas, utilizaremos las de **clasificación**, **evolutiva** y de **relación con el hardware**.

Perspectiva de clasificación

Una forma de establecer la relación existente entre paradigmas es la creación de una clasificación de los mismos.

Una de las posibles clasificaciones tiene que ver con el hecho de que un paradigma es, a su vez, una clasificación de lenguajes de programación. Por tanto, agruparemos los paradigmas según sea **la clasificación** que establecen **sobre los lenguajes de programación**.

Algunos paradigmas establecen, cada uno de ellos, una partición sobre el total de los lenguajes de programación. La figura 2.3 representa esta idea incluyendo el conjunto de todos los posibles lenguajes de programación, representados por la elipse, donde tenemos dos grupos definidos, establecidos por el cumplimiento o no del paradigma de la programación estructurada.

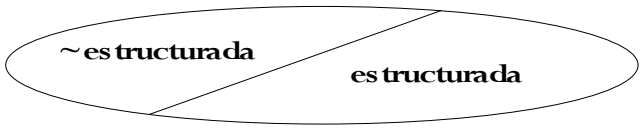

Figura 2.4: Paradigma de clasificación de todos los lenguajes

El nombre que daremos a estos paradigmas, es el de **Paradigmas generales**.

La relevancia de estos paradigmas para la clasificación de lenguajes es muy elevada y, tanto el hecho de cumplir, como de no cumplir tienen importancia.

Un segundo tipo de paradigma según la clasificación que hace de los lenguajes son los **Paradigmas complementarios**. Se trata de un par de paradigmas con la particularidad de que no cumplir uno implica cumplir el otro. Un ejemplo muy claro de este tipo de paradigmas son el paradigma imperativo y el declarativo. Estos paradigmas también establecen una partición total del conjunto de los lenguajes y se solapará con las particiones establecidas mediante los paradigmas generales.

Existe un tercer tipo de paradigma muy parecido a los paradigmas generales que denominaremos **Paradigmas específicos** que establecerán una partición del mismo tipo que los generales pero en los que el hecho de no cumplir el paradigma no tendrá relevancia ninguna para la caracterización de un lenguaje. Por tanto, tendremos la misma separación, en dos grupos, pero para el grupo que no cumple el paradigma, se ignora este hecho por su irrelevancia.

Todas estas clasificaciones se solapan ya que todas afectan al conjunto de los lenguajes. Por tanto, las podemos contemplar con este solapamiento como vemos en la Figura 2.4.

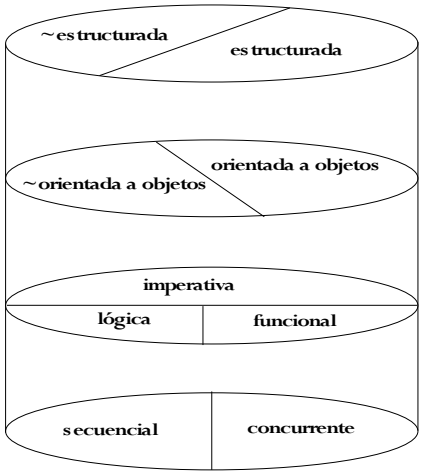

Figura 2.5: Clasificaciones de lenguajes solapadas

En esta Figura tenemos paradigmas generales como el estructurado y el orientado a objetos, paradigmas excluyentes como el imperativo y el declarativo y específicos como el de tiempo real.

Además de esta **clasificación** podemos realiza otra **según el tipo de restricciones que** el paradigma **establece en los lenguajes** que lo cumplen.

Es una clasificación global en dos grandes categorías: **paradigmas orientados a establecer la misión del lenguaje** y **paradigmas orientados a establecer la forma que deberán tener los programas** de un determinado lenguaje para cumplir dicho paradigma y, desde luego, obtener los beneficios que dicho paradigma consigue.

Llamaremos a los primeros "**Paradigmas de Objetivo**" y a los segundos "**Paradigmas de Estructura**".

Dentro de los Paradigmas de Objetivo tendremos:

- Programación Concurrente.
- Programación Declarativa.
- Programación Distribuida.
- Programación en Tiempo Real.
- Programación Funcional.
- Programación Imperativa.
- Programación Lógica.

Y para los Paradigmas de Estructura tendremos:

- Programación Estructurada.
- Programación Modular.
- Programación Orientada a Módulos.
- Programación Orientada a Objetos.
- Programación Orientada a Procedimientos.

Dentro de esta primera separación tendremos otras más complejas que iremos desglosando en los siguientes apartados.

Paradigmas de objetivo

Estos paradigmas se caracterizan por establecer, para un lenguaje, el significado de sus sentencias, no sólo en su traducción a un código máquina de un microprocesador sino, también, en su representatividad de conceptos modelados en la fase de diseño.

Los programas creados con lenguajes que cumplen algunos de estos paradigmas tendrán unas capacidades no disponibles en el resto. Un ejemplo de este tipo de paradigmas es el de tiempo real, que describe unas capacidades que no forman parte de ningún otro paradigma por no ser necesarias para ninguna otra forma de programar.

Paradigmas de estructura

Estos paradigmas se caracterizan por establecer, para un lenguaje, unas ciertas recomendaciones de estructura de programas para evitar algunos problemas inherentes u obtener unas algunas ventajas del lenguaje y que no serán evitadas u obtenidas de forma automática o necesaria en todo programa desarrollado utilizando el lenguaje. La realización de los programas según el paradigma, es opcional en algunos casos y obligatoria en otros, dependiendo del lenguaje, pero su seguimiento garantiza las características que dicho paradigma propugna.

Perspectiva evolutiva

La aparición de los paradigmas se ha ido produciendo en el tiempo a medida que los lenguajes de programación se desarrollaban y mejoraban en sus características. Algunos paradigmas se pueden considerar basados en otros o contrapuestos, habiendo surgido unos por las necesidades que generaban, o dejaban de cubrir, otros.

Estudiar los paradigmas siguiendo este enfoque es la perspectiva evolutiva y tenemos como secuencia más significativa la aparición del paradigma imperativo seguida del declarativo como contrapuesto y la posterior evolución de ambos desde el paradigma puro sin organización a paradigma estructurado, después a orientado a objetos y ahora a orientado a aspectos (este último no del todo consolidado y en competencia con otras tendencias).

Otra línea significativa es la de la mejora del paradigma secuencial con la aparición del paradigma concurrente con las dos vertientes del mismo en programación para memoria compartida y distribuida, dentro de los cuales tenemos posteriores refinamientos como cliente/servidor y otros.

Los lenguajes evolucionan y cambian, aparecen lenguajes y otros desaparecen o permanecen con uso muy reducido, pero algunos marcan un nuevo escalón en el avance de esta ciencia. Estos lenguajes son los que proponen un nuevo paradigma, los que dan un paso en la evolución real de la programación.

La evolución de los paradigmas es la evolución de la programación.

Perspectiva de relación con el hardware

El último enfoque de estudio de los paradigmas por el que nos interesamos es el de su relación con el hardware. Ya que todo lo que se escriba en un programa debe ser realizado por el HW sobre el que se ejecuta, **los lenguajes de programación son siempre abstracciones del HW** para mayor comodidad del programador.

Con estas premisas, podemos clasificar los paradigmas en función de su **cercanía al HW**. Esta cercanía no la contemplaremos como en los lenguajes de programación en función de lo distintas que sean las instrucciones sino según lo parecido o distinto que sea el enfoque aplicable a la resolución de los problemas.

Los paradigmas que ofrecen una visión radicalmente distinta en el enfoque son el imperativo y el declarativo. El primero es una aplicación directa de la arquitectura subyacente y el segundo es un alejamiento que elimina de los lenguajes algunas de las características disponibles en el HW.

Otros ejemplos pueden ser el paradigma estructurado y OO ya que el HW no trabaja con los conceptos que manejan estos PdP. Si además los combinamos con paralelismo en un HW no paralelo o con el paradigma declarativo, vemos que pueden aparecer muchos niveles distintos de alejamiento o cercanía. Conocer estos niveles puede ser útil de cara a elegir lenguajes en proyectos ligados a requerimientos HW específicos o con necesidades de procesar datos más cercanos al ser humano.

Descripción de algunos PdP

A continuación presentamos la descripción de algunos de los principales ejemplos de

paradigmas como orientación que ayude a la comprensión general de las ideas en torno a los paradigmas.

Programación imperativa

Los lenguajes imperativos están orientados a la acción, en el estilo de la máquina de Von Neumann, es decir, contemplan la ejecución de una acción como una secuencia de operaciones sobre posiciones de memoria.

Los programas escritos con este tipo de lenguajes describen "cómo" deben obtener los resultados de la máquina por medio de órdenes directas a ejecutar sobre ésta. Las ordenes siempre son sentencias de asignación (cambiar el estado de una variable en memoria por un nuevo valor calculado) y sentencias de control de flujo (cambios en la secuencia de ejecución de las asignaciones).

Estos lenguajes son considerados eficientes y con una notación relativamente cercana al lenguaje natural, debido a que su sintaxis se crea mediante combinación de palabras en lengua inglesa, identificadores elegidos por el programador y signos comunes como los operadores aritméticos o los paréntesis.

Los ejemplos más notables de lenguajes imperativos a lo largo de la historia son: Fortran (1957), Algol (1960), Pascal (1971), C (1972), C++ (1985), o Java (1995).

Programación declarativa

La programación declarativa se diferencia de la programación imperativa en que los programas declarativos describen el "qué" se desea obtener. Los lenguajes que cumplen con la programación declarativa, se pueden dividir en dos, los lenguajes funcionales y los lenguajes lógicos.

Todos los lenguajes declarativos, se caracterizan por la expresión en un programa de lo que se desea conseguir sin especificar cómo debe hacerse.

Pero la notación y el alcance de los programas que se crean con los dos paradigmas que incluye, puede cambiar bastante. En el caso de la programación funcional, las tareas a realizar se modelan mediante funciones y expresiones, con ausencia de sentencias de procedimientos explícitos. En el caso de la programación lógica, tendremos hechos y relaciones entre ellos como medio para representar las tareas que se quieren realizar.

La programación funcional se ha considerado ineficiente y complicada desde sus orígenes debido al bajo rendimiento de las primeras implementaciones de los lenguajes. En cambio, tenemos la ventaja de la gran expresividad, la extensibilidad, la facilidad de corrección y la falta de efectos laterales.

> El primer lenguaje de programación funcional fue LISP (McCarthy— 1959), que se basa en el procesamiento de listas, le siguieron APL (1962), ISWIM (1966), SCHEME (1975), FP (1977), HOPE (1980), MIRANDA (1985), ML (1986), HASKELL (1988),....

La programación lógica, basada en el razonamiento lógico, se usa para crear sistemas que tratan responder a las demandas del usuario, manejando relaciones entre datos (hechos) en vez de funciones

Un ejemplo de programación lógica puede ser:

```
(hecho) Cualquier psiquiatra es una persona
(hecho) La persona a la que analiza está enferma
(hecho) Juan es un psiquiatra de Madrid
(pregunta) →¿Es Juan una persona?
(pregunta) →¿Dónde está Juan?
(pregunta) →¿Está Juan enfermo?
```

El primer lenguaje de programación lógica y el más exitoso en sus muchas versiones es PROLOG (1972)

Programación concurrente

Este paradigma hace referencia a la creación de programas que sean capaces, por su programación, de realizar simultáneamente varias de las tareas que componen el algoritmo que implementan.

Para que puedan existir son necesarios algoritmos con unas ciertas características que permitan crear estos programas obteniendo el mismo resultado que si la ejecución fuese secuencial pero con mayor rendimiento.

También será preciso el uso de técnicas, lenguajes, librerías y hardware específico, para conseguir el paralelismo.

Existen dos enfoques distintos de paralelismo, la programación concurrente de memoria compartida y la programación concurrente distribuida.

Programación de memoria común

Se basa en la existencia de múltiples tareas de ejecución simultánea sobre una única máquina con una única memoria. La existencia de varios procesadores no será un requisito imprescindible pero si deseable. La característica de la memoria única condicionará la arquitectura de los programas, las necesidades, las posibilidades y los problemas.

Programación distribuida

En este caso, se supera la barrera de la máquina única y la memoria única. Se considerará que los procesos deben ejecutarse en sistemas distintos, distribuyendo el trabajo a través de la necesaria conexión entre las máquinas. El uso de estas líneas de comunicación será, en este caso, el que va a condicionar la forma en que se crean los programas.

Programación orientada a objetos

Este paradigma es una evolución del Paradigma Estructurado consistente en extender el mismo mediante la fragmentación del código estructurado y la reunión de los fragmentos que trabajan sobre un cierto dato en torno al mismo.

Con esta reunión se crea el concepto de Objeto, entidad central del paradigma. Estos objetos son tipos de datos que engloban bajo el mismo nombre las características (estructuras de datos) del objeto del mundo real que modelan (atributos) y los algoritmos para manipular esas características (métodos).

Las principales ventajas de este paradigma son:

- Los mecanismo de encapsulación permiten un alto grado de reutilización del código (se incrementa con la herencia).
- Permite la representación más directa del mundo real en el código (abstracción).
- Se adapta mejor a la informática distribuida y a los modelos cliente/servidor.
- Se pueden crear desarrollos más flexibles y el proceso es más rápido.

El primer lenguaje de programación orientado a objetos fue Simula (1967) y le siguieron Smalltalk (1980), Eiffel (1988), existiendo en la actualidad muchos lenguajes orientados a objetos, siendo los más exitosos los lenguajes híbridos como versiones de: C++, Objetive C, CLOS, Ada o Java.

Programación de tiempo real

Este paradigma proporciona las técnicas para controlar el funcionamiento de sistemas bajo ciertas restricciones de tiempo.

Estas restricciones temporales obligarán a tener un control permanente y simultáneo sobre la ejecución de varias de las funcionalidades de los programas, siendo preciso, en ocasiones, interrumpir una para cumplir las restricciones de ejecución de otra.

La programación concurrente suele incorporar características de este paradigma, pero la utilización de uno de los dos PdP no implica la utilización del otro. Esto va en contra de la idea generalizada de que la programación de tiempo real es un tipo de programación concurrente, que es incorrecta.

Los programas de tiempo real suelen instalarse en un ordenador que recibe información de la situación externa, una vez procesados los datos que llegan al ordenador, se envía una respuesta (comando de control para que el sistema reaccione ante el cambio de estado que se ha producido). El tiempo entre la petición y la respuesta debe cumplir los plazos establecidos para el sistema en todos los casos.

Los lenguajes que soportan este paradigma permiten tratar las interrupciones (tanto internas como externas) y acceder a las interfaces de E/S, como único medio posible de garantizar el cumplimiento de las restricciones temporales.

En cuanto a la posibilidad de utilizar programación distribuida que tenga las características de la programación de tiempo real, no hay una postura clara a adoptar ya que las restricciones temporales no se podrán asegurar si se basan en un hardware, la red, con una fiabilidad baja. Sin embargo, en ciertas circunstancias o configuraciones esto sí podría ser posible con una red redundante, segura y rápida.

EJERCICIOS

1.- Establecer el nivel de cumplimiento del paradigma estructurado de los siguientes lenguajes de programación:

- Pascal
- C
- C++
- Java
- Modula-2
- Ada
- Fortran
- Basic
- Caml

2.- Establecer los paradigmas que cumplen y el nivel de cumplimiento para los siguientes lenguajes de programación (consultando bibliografía o realizando búsquedas en Internet):

- Pascal
- C
- C++
- Cobol
- Parlog
- Caml
- Algol
- Java
- Modula-2
- Ada
- Lisp
- Modula-2
- OCaml
- Ensamblador
- Fortran
- Basic
- Caml
- Prolog
- Ada
- SmallTalk
- CLOS

3.- Localizar la mayor cantidad posible de lenguajes de programación, distintos de los anteriores, y establecer sus características mediante el cumplimiento de paradigmas de cada uno de ellos. De nuevo se utilizará la bibliografía y la consulta a través de Intenet.