

Lenguajes de Programación Multi-paradigma

Diseño y Paradigmas de Lenguajes¹

¹Departamento de Informática
Universidad Nacional de San Luis, Argentina



Octubre de 2015

Conceptos Generales

¿Qué es un **paradigma de programación**?

- Es un enfoque para programar una computadora basado sobre una **teoría matemática** o un **conjunto coherente de principios**.
- El conjunto de conceptos que soporta un paradigma, lo hace más adecuado para una cierta **clase de problemas**.

Conceptos Generales

¿Qué es un **paradigma de programación**?

- Es un enfoque para programar una computadora basado sobre una **teoría matemática** o un **conjunto coherente de principios**.
- El conjunto de conceptos que soporta un paradigma, lo hace más adecuado para una cierta **clase de problemas**.

Ejemplos:

- POO es la mejor elección para problemas con un gran número de abstracciones de datos relacionados, organizados en una jerarquía.
- Programación Lógica soporta de manera natural la transformación y/o análisis de estructuras simbólicas complejas de acuerdo a **reglas lógicas**.

Conceptos Generales

Principales **paradigmas de programación**

- Programación **Imperativa**
- Programación **Declarativa**
 - Programación **Lógica**
 - Programación **Funcional**
- Programación **Orientada a Objetos**
- Programación **Concurrente**
- Programación con **Restricciones**
- Programación basada en **Eventos** (Reactiva)
- **Meta-programación**
- otros (**Visual**, basada en **Aspectos**, etc)

Conceptos Generales

¿Qué es un **lenguaje de programación multi-paradigma (LPMP)**?

Es un lenguaje de programación que soporta **más de un paradigma** de programación.

Conceptos Generales

¿Qué es un **lenguaje de programación multi-paradigma (LPMP)**?

Es un lenguaje de programación que soporta **más de un paradigma** de programación.

¿Por qué son **útiles** los LPMPs?

- Problemas de programación realistas, necesitan diferentes **conceptos de programación**.
- Para resolverlos de forma clara y simple, uno o dos paradigmas (como es usual en los lenguajes más difundidos) no son suficientes.

Soporte para programación multi-paradigma

Lenguajes con “extensiones” multi-paradigma

Extienden un **paradigma principal**, con características propias de **otro paradigma**. Ejemplos:

- C++ (extiende Imperativo con POO)
- Prolog (extiende Programación Lógica con imperativo)
- CLOS (extiende Programación Funcional con POO)

Soporte para programación multi-paradigma

Lenguajes con “extensiones” multi-paradigma

Extienden un **paradigma principal**, con características propias de **otro paradigma**. Ejemplos:

- C++ (extiende Imperativo con POO)
- Prolog (extiende Programación Lógica con imperativo)
- CLOS (extiende Programación Funcional con POO)

Lenguajes Multi-paradigma “reales”

Son diseñados para proveer un real marco multi-paradigma al programador de un sistema. Ejemplos:

- Oz
- Alice
- Leda
- Ciao

Programación Imperativa

Paradigma que describe la computación en términos de **sentencias/instrucciones/órdenes** que cambian el **estado** de un programa.

Los programas imperativos definen **secuencias** explícitas de **comandos** que la computadora debe ejecutar.

Ejemplos de lenguajes imperativos

- FORTRAN
- COBOL
- BASIC
- C

Programación Imperativa (II)

Es el paradigma que refleja más directamente el funcionamiento del **hardware tradicional** de las computadoras.

Muchos lenguajes imperativos, pueden ser considerados **abstracciones** del lenguaje assembly y el lenguaje de máquina.

La instrucción por excelencia es aquella que modifica el estado del programa (contenido de la memoria) como por ejemplo, la **asignación de variables**

Programación Funcional

Paradigma que describe la computación en términos de **funciones** que son aplicadas a datos, y cuyos resultados sirven como argumentos para otras funciones (**composición de funciones**).

La metáfora subyacente es la evaluación de **funciones matemáticas**, es decir, funciones **sin efectos colaterales** y cuyos resultados sólo dependen de los argumentos explícitos de entrada.

Ejemplos de lenguajes funcionales

- FP (John Backus)
- Lisp (John McCarthy)
- Haskell

Programación Funcional (II)

Este paradigma toma como marco teórico al “**lambda calculus**”, una abstracción matemática que permite describir funciones y su evaluación.

Especial énfasis en evitar la idea de **estado** de una computadora (variables) y su cambio a lo largo del tiempo

Funciones siguen la idea de **expresiones**, es decir no hay **efectos colaterales** en memoria o I/O

Lenguajes imperativos también soportan funciones pero no en el sentido matemático, sino de **subrutinas**. Efectos colaterales pueden cambiar el estado del programa, y de esa manera se pierde **transparencia referencial**.

Programación Funcional (III)

Algunas características “típicas” en el paradigma funcional:

- Las funciones son “objetos” de **primera clase**, que son tratados como un “dato” o “valor” más (por ejemplo, pasados como argumentos en iteradores, filtros, etc). Incluso puede que no tengan asociado un nombre (funciones lambda), como puede hacerlo un valor numérico común.
- La **recursión** es la estructura de control primaria.
- Especial énfasis en el procesamiento de **listas**.
- Funciones evitan **efectos de lados**.
- Evaluación de **expresiones** en lugar de usar **sentencias**.

Programación Orientada a Objetos

Paradigma que describe la computación en términos de **objetos** que se comunican mediante el envío de **mensajes**.

Un lenguaje orientado a objetos debe proveer soporte para:

- Tipos de Datos Abstractos.
- Herencia.
- Ligadura dinámica de mensajes a métodos.

Ejemplos de lenguajes orientados a objetos

- Smalltalk
- Objective-C
- Java

Programación Orientada a Objetos (II)

La POO tiene sus raíces en SIMULA-67, un lenguaje que ya en el año 1967 introduce conceptos como clases, objetos, herencia, etc.

Los conceptos de la POO no fueron completamente desarrollados hasta el surgimiento de SMALLTALK 80, para muchos, el único lenguaje de POO “puro”.

Contribución fundamental de la POO: **polimorfismo** basado en los siguientes mecanismos:

- **ligadura dinámica** de mensajes a definiciones de métodos.
- **variables polimórficas**.

Programación Lógica

Paradigma de programación basado en la **lógica formal**, que visualiza a un programa de computadora como un conjunto de **sentencias lógicas** (cláusulas) que describen **hechos** y **reglas** sobre algún dominio de un problema.

En este contexto, un **algoritmo** estará implementado por un programa lógico, más un **mecanismo de control** (probador de teoremas) que realiza **inferencias** (**razona**) con las sentencias del programa.

Ejemplos de lenguajes de programación lógica

- Prolog
- Datalog
- Answer Set Programming (ASP)
- DeLP (Defeasible Logic Programming)

Programación Lógica (II)

Algunas formas restringidas de sentencias, como las **cláusulas de Horn** (disjunciones con a los más un literal positivo) permiten realizar **inferencia** (hacia **adelante** o **atrás**) de manera **natural** y **eficiente**.

Prolog surge como el lenguaje de programación lógica más usado con aplicaciones en:

- “prototipeado” rápido
- tareas de manipulación de símbolos (escritura de compiladores, parsing de lenguaje natural)
- sistemas expertos (legales, médicos, financieros)

Extensiones son posibles para el manejo de **restricciones** y el **razonamiento no monótono**.

Programación Concurrente

Paradigma de programación que soporta la idea de tener **varias computaciones (actividades)** ejecutándose en períodos de tiempo que se solapan.

Cada actividad tiene un **flujo de control separado**, que se puede dar entre **computadoras** (sistemas distribuidos), **procesos** (con memorias independientes) en una computadora, o **hilos (threads)** dentro de un proceso que comparten el mismo espacio de memoria.

Programación Concurrente (II)

Los lenguajes de programación concurrente proveen construcciones que soportan conceptos como “**multi-threading**”, computación distribuida, **pasaje de mensajes**, recursos compartidos (incluyendo **memoria compartida**) y “**futures**” y **promesas**.

Ejemplos de lenguajes con programación concurrente

- Java
- Erlang
- Concurrent C
- Ada
- C#

Programación Concurrente (III)

Aspecto clave: modelo de **comunicación/interacción**

- Implícita (“futures” and promises)
- Explícita
 - Memoria compartida (ej. Java y C#). Usual/ proveen mecanismos de **bloqueado**/“locking” (**monitores/semáforos/mutex**)
 - Pasaje de mensaje (síncrono o asíncrono)

Buena integración con **paradigmas declarativos**

- Programación Lógica Concurrente
- Programación Lógica con Restricciones Concurrente
- Programación Funcional Concurrente (ver artículo “The Downfall of Imperative Programming”)

Programación con Restricciones

Paradigma de programación que se basa en la especificación de **restricciones** que un conjunto de variables debe cumplir.

Es una forma de **programación declarativa**, ya que especifica propiedades que la solución debe cumplir, y no los pasos que se deben realizar para encontrarla.

Usual/ se integra a un **lenguaje huésped**. Enfoques:

- Embebido en el lenguaje
- Bibliotecas de software separadas

Por consiguiente, es inherentemente multi-paradigma:

- Programación lógica con restricciones
- Programación funcional con restricciones
- Programación imperativa con restricciones

Programación con Restricciones (II)

Ejemplos de lenguajes que soportan restricciones

- Prolog III
- Oz
- Kaleidoscope

Estos lenguajes deben proveer una forma de especificar las componentes de un **problema de satisfacción de restricciones**:

- 1 Un conjunto de **variables**
- 2 El **dominio** para cada variable
- 3 Las **restricciones** que las variables deben cumplir.

Programación con restricciones (III)

Ejemplo: SEND+MORE=MONEY en Prolog (+ CLPFD)

```
:- use_module(library(clpfd)).
sendmore(Digits) :-
    Digits = [S,E,N,D,M,O,R,Y],
    Digits ins 0..9,
    S #\= 0,
    M #\= 0,
    all_different(Digits),
    1000*S + 100*E + 10*N + D
    + 1000*M + 100*O + 10*R + E
    #= 10000*M + 1000*O + 100*N + 10*E + Y,
    label(Digits).
```

Programación con restricciones (IV)

Resultado de la consulta:

```
?- sendmore(X) .
```

```
X = [9, 5, 6, 7, 1, 0, 8, 2]
```

Metaprogramación

La metaprogramación es la capacidad de escribir programas de computadora que pueden tratar **programas** como si fueran **datos** comunes.

En otras palabras, un programa escrito en un lenguaje que provee metaprogramación podría leer, generar, analizar y/o transformar otros programas, e incluso a sí mismo, durante su **ejecución**.

Enfoques usuales en metaprogramación:

- Metaprogramación de “templates”
- Reflexión/reflectividad

Reflexión (reflectividad)

Habilidad de un programa para **examinar** (y por lo tanto **razonar**) sobre su propio estado interno (**introspección**) y/o **modificar** la **estructura** y **comportamiento** del programa (**"intercession"**) en **tiempo de ejecución**.

Característica usual en lenguajes assembly (instrucciones definidas como datos y código auto-modificadorio). Luego, se extiende a lenguajes interpretados de los paradigmas lógico, funcional y orientado a objetos.

Reflexión (reflectividad) (II)

Algunas **habilidades reflectivas** usualmente provistas en tiempo de ejecución:

- Descubrir y modificar construcciones de código fuente (bloques de código, clases, métodos, protocolos, etc) como un objeto de primera clase en tiempo de ejecución.
- Convertir un string que corresponde al nombre simbólico de una clase o función en una **referencia a** o una **invocación de** dicha clase o función.
- Evaluar un string como si fuera una sentencia de código fuente en tiempo de ejecución.
- Crear un nuevo intérprete para el bytecode (instrucciones de la máquina virtual) del lenguaje para dar un nuevo significado o propósito para una construcción de programación.

Reflexión en SmallTalk

Introspección

- `class`. Pregunta a una instancia la clase a la que pertenece.
- `allInstances`. Consulta a una clase todas sus instancias.
- `instVarNames`. Dá los nombres de todas las variables de instancia de una clase.
- `selectors`. Dá los nombres de todos los métodos implementados por una clase.
- `getSource`. Dado un método compilado (objeto) retorna su código fuente.

Reflexión en SmallTalk

“Intercession”

- `<class> compile: <text> classified:
<category>`. **Agrega dinámicamente un nuevo método a alguna clase.**
- También es posible agregar subclases **dinámicamente**

Mozart - Oz

Mozart es una implementación de Oz, un lenguaje de programación multi-paradigma que soporta los estilos:

- declarativo
- funcional
- orientado a objetos
- concurrente
- distribuido
- lógico
- programación con restricciones

como partes de una totalidad **coherente**.

`www.mozart-oz.org`

Alice

Basado en el lenguaje ML standard, provee soporte para:

- programación funcional
- programación concurrente
- programación distribuida
- programación con restricciones

`www.ps.uni-sb.de/alice`

Leda

Con una sintaxis influenciada por la del lenguaje de programación ALGOL, las técnicas soportadas por Leda incluyen:

- Programación imperativa
- el enfoque orientado a objetos
- programación lógica
- programación funcional

courses.cs.vt.edu/~cs5314/Lang-Paper-Presentation/Papers/HoldPapers/LEDA.pdf

Ciao

Tomando com referencia una variante del Prolog (&-Prolog), este lenguaje multi-paradigma provee soporte para:

- Programación lógica
- Programación funcional
- Programación con restricciones
- Programación Orientada a Objetos
- Concurrencia, paralelismo y ejecución distribuida

clip.dia.fi.upm.es/papers/hermenegildo08:ciao-design-ugo-fest.pdf