# Dimensiones de diseño

Los DSLs son herramientas poderosas en la ingeniería de software, porque ante un problema en particular ofrecen una solución a la medida. El diseño de un DSL viene con un alto grado de libertad, que hace díficil encontrar un nivel correcto de abstración, una cobertura adecuada del dominio en estudio y consistencia entre elementos.

Para enfrentar dichos retos, Voelter propone un "framework" para describir y caracterizar los DSL. Dicho "framework" identifica 7 dimensiones, para un correcto diseño de un DSL:

1. Expresividad
2. Cobertura
3. Semánticas
4. Separación de preocupaciones
5. Completitud
6. Modularización del lenguaje
7. Sintaxis.

## Expresividad

Los DSLs, son mucho más expresivos que los GPLs. Como consecuencia de esto, se obtiene programas más cortos y con mayor disponibilidad para procesamiento de la semántica por parte de las herramientas (debido a que los programas cortos son más fáciles de escribir y leer, son considerados más eficientes dentro de la ingeniería de software).

Por tanto, los DSLs son más abstractos que los GPLS, ya que evitan describir detalles que son irrelevantes al propósito del modelo.

### "Linguistic Abstractions versus In-Language Abstractions"

Existen dos formas para definir abstracciones. Las abstracciones pueden construirse dentro del lenguaje ("*linguistic abstractions*"), o pueden ser expresadas por conceptos disponibles dentro del lenguajes ("*in-language abstractions*"). Los DSLs por lo general, se apoyan en *linguistic abstractions*, y los GPLS se apoyan: "*in-language abstractions*.

"**Linguistic abstraction**";, un concepto de dominio específico puede ser modelado con la ayuda de una abstracción existente, o con la adición de una nueva abstracción para representar dicho concepto.

No hay necesidad de considerar una relevancia en la semántica o patrones para expresar programas dentro de un dominio en particular. Esto se puede ejemplificar con los siguientes fragmentos de código:

1.

int[] arr = ... for (int i = 0; i<arr.size(); i++) { sum += arr[i]; }

2.

int[] arr = ... OrderedList<int> l = ... for (int i=0; i<arr.size(); i++) { l.add( arr[i] ); }

En el primer ciclo, el orden en que se procesan los elementos es irrelevante, por tanto, se puede paralelizar. El segundo ciclo, no se puede paralelizar, debido a que el orden de los elementos en la clase OrderedList es relevante.

En el segundo ciclo, se necesita un motor de transformación que traduzca y optimice este tipo de programas, además se deber realizar un análisis sofisticado del programa para determinar si existe la posibilidad de paralelizar dicho fragmento de código.

La siguiente expresión, presenta una alternativa del mismo comportamiento. Se utiliza un "*linguitic abstraction*" para indicar que no hay necesidad de análisis para determinar que el primer ciclo puede paralelizarse y el segundo no.

for (int i in arr) { sum += i; }

seqfor (int i in arr) { l.add( arr[i] ); }

El segundo ciclo **declara** que no puede ser paralelizado. *Declaratividad* es la propiedad de un lenguaje de dominio, que tiene conceptos de primera clase para abstracciones relevantes dentro de un dominio. No se necesita un sofisticado mecanismo de "pattern matching" o un análisis del flujo de programa para capturar la semántica de un programa.

### "In-Language Abstraction"

Utiliza conceptos bien conocidos de los GPLs, para construir nuevas abstracciones (procedimientos, clases, funciones, genéricos, traits y monads).

**No es señal de un mal DSL** si se tienen mecanismo de abstracción dentro del lenguaje, siempre y cuando dichas abstracciones no requieran un tratamiento especial en las herramientas de análisis y procesamiento. Un ejemplo de un tratamiento especial, sería si el compilador sabe que la librería de clases de OrderedList es ordenada, y por tanto conoce que un ciclo que utilice dicha clase no puede ser paralelizado.

En "in-language abstraction" existe abstracción, pero no existe *declaratividad*: Una herramienta que procese el modelo, tiene que entender la intención del usuario al momento de analizar la abstracción.

Un alternativa, es la definición de una librería estandar. En lugar de adicionar nuevos aspectos al lenguaje, una librería estandar, puede exponer abstracciones relevantes a sus usuarios (expresadas como si fueran "*in-language abstractions*"). Este enfoque, permite que el lenguaje como tal, permanezca pequeño y que las futuras extensiones de la librería, no cambien la definición del lenguaje o las herramientas de procesamiento.

## Cobertura

Un lenguaje L da una total cobertura a un dominio D, si para cada programa pD relevante dentro del dominio D, existen un programa pL equivalente, que puede ser escrito por medio del lenguaje L.

La cobertura total, es un predicado "*Boolean*". Un lenguaje cubre en su totalidad un dominio, o no lo hace. En la práctica, muchos lenguajes no cubren en su totalidad un dominio, por tanto se ha definido un *radio de cobertura*. El radio de cobertura de un lenguaje L, es la porción de programación en el domino D que pueden expresarse.

radio de cobertura = número de programas que se pueden expresar por L / el número de programas dentro del dominio D

A simple vista, un DSL idea debe cubrir todo su dominio en un 100%. Sin embargo, se requiere que el dominio este bien definido y se necesita conocer lo que realmente es una cobertura total. Además, en el transcurso del tiempo, es posible que el dominio evolucione y crezca, y que el lenguaje tenga que evolucionar constantemente para mantener la cobertura total.

Además de la evolución, existen dos razones para que un DSL no cubra, totalmente un dominio:

1. El lenguaje, puede ser deficiente y necesite ser rediseñado. Esto es especialmente cierto, para DSLs nuevos e inmaduros. Deliminar el dominio para el cual es contruído el DSL, es especialmente importante.
2. El lenguaje en su origen, pudo haberse definido exclusivamente para un subconjunto del dominio, por lo general dicho subconjunto es el más usado. La cobertura de todo el dominio, puede llevar a que el lenguaje sea muy grande o complicaod para el usuario final; esto es debido al poco soporte a los casos de borde poco usado dentro del dominio. Estos casos de borde, por lo general deberían ser expresados por un GPL.