Abordaje Funcional a EDSLs

Alberto Pardo Marcos Viera

Instituto de Computación, Facultad de Ingeniería
Universidad de la República, Uruguay

ECI 2024

Embedded Domain Specific Languages

Domain Specific Languages (1)

Un **lenguaje de dominio específico** (DSL) es un lenguaje de programación o especificación de expresividad limitada, especialmente diseñado para resolver problemas en un particular dominio.

Ejemplos:

- HTML
- VHDL (hardware)
- Mathematica, Maple
- SQL,XQuery (lenguajes de query)
- Yacc y Lex (para la generación de parsers)
- LaTeX (para producir documentos)
- DSLs para apl. financieras (`http://www.dslfin.org`)

Domain Specific Languages (2)

Existen dos abordajes principales para implementar DSLs:

Externo: lenguaje standalone

Es necesario desarrollar:

- lexer, parser
- compilador
- herramientas

Interno: lenguaje implementado en el contexto de otro (*embebido*)

DSLs embebidos

- Embedded DSLs (EDSLs) son DSLs implementados como bibliotecas específicas en lenguajes de propósito general que actuan como anfitrión (*host languages*)
- De esta manera el EDSL puede hacer uso de la infraestructura y facilidades existentes en el lenguaje anfitrión.
- La implementacion de un EDSL suele reducir el costo de desarrollo (se evita implementar lexer, parser, etc).
- Los lenguajes funcionales, en particular Haskell, son muy apropiados para la implementación de EDSLs.
- El manejo de errores suele ser un punto debil de los EDSLs.

Ejemplos de EDSLs

Algunos ejemplos de EDSLs en Haskell:

- QuickCheck
- Sequence (finger trees)
- Streams
- HaXml (procesamiento de XML, HTML)
- Lava (hardware description)
- Parsec (parsing)
- Pretty printing
- Haskore (para componer música)

Tipos de EDSLs

- **Shallow embedding**
  - Se captura directamente en un tipo de dato la semántica de los datos del dominio.
  - Dicha interpretación es fija.
  - Las operaciones del DSL manipulan directamente los valores del dominio.

Tipos de EDSLs

- **Shallow embedding**
  - Se captura directamente en un tipo de dato la semántica de los datos del dominio.
  - Dicha interpretación es fija.
  - Las operaciones del DSL manipulan directamente los valores del dominio.

- **Deep embedding**
  - Las construcciones del DSL son representadas como términos de tipos de datos que corresponden a árboles de sintaxis abstracta (AST).
  - Estos términos son luego recorridos para su evaluación.
  - No hay una semántica fija, sino que se pueden definir diferentes interpretaciones.

Ejemplo de EDSL

Consideremos un lenguaje que manipula expresiones aritméticas formado por las siguentes operaciones:

$$
\begin{array}{lll}
val :: Int \to Expr & & \text{-- constructor} \\
add :: Expr \to Expr \to Expr & & \text{-- constructor} \\
eval :: Expr \to Int & & \text{-- observador}
\end{array}
$$

Ejemplo de EDSL

Consideremos un lenguaje que manipula expresiones aritméticas formado por las siguentes operaciones:

$$
\begin{array}{lll}
val :: Int \to Expr & & \text{-- constructor} \\
add :: Expr \to Expr \to Expr & & \text{-- constructor} \\
eval :: Expr \to Int & & \text{-- observador}
\end{array}
$$

Ejemplo de un programa en el DSL:

$$
\begin{array}{l}
siete :: Expr \\
siete = add\ (val\ 3)\ (val\ 4)
\end{array}
$$

$$
\begin{array}{l}
doble :: Expr \to Expr \\
doble\ e = add\ e\ e
\end{array}
$$

$$
\begin{array}{l}
runDoble :: Expr \to Int \\
runDoble\ e = eval\ (doble\ e)
\end{array}
$$

Shallow embedding

Se captura directamente la semántica del dominio que manipula el DSL.

Para este tipo de expresiones aritméticas la representación por defecto es usar un entero, el cuál va a denotar el valor de la expresión.

$$\texttt{type}\ \mathit{Expr} = \mathit{Int}$$

Shallow embedding

Se captura directamente la semántica del dominio que manipula el DSL.

Para este tipo de expresiones aritméticas la representación por defecto es usar un entero, el cuál va a denotar el valor de la expresión.

$$\texttt{type}\ \mathit{Expr} = \mathit{Int}$$

Constructores

$$\begin{aligned} \mathit{val}\ n &= n \\ \mathit{add}\ e\ e' &= e + e' \end{aligned}$$

Observador

$$\mathit{eval}\ e = e$$

Deep embedding (1)

Se definen las formas de construir expresiones a través de un tipo.

```
data Expr where
  Val  :: Int → Expr
  Add :: Expr → Expr → Expr
```

Deep embedding (1)

Se definen las formas de construir expresiones a través de un tipo.

```
data Expr where
  Val  :: Int → Expr
  Add :: Expr → Expr → Expr
```

Operaciones de construcción (smart constructors):

```
val :: Int → Expr
val n = Val n

add :: Expr → Expr → Expr
add e e' = Add e e'
```

Deep embedding (2)

El **observador** ahora hace las veces de función de interpretación.

$$
\begin{aligned}
&\mathit{eval} :: \mathit{Expr} \to \mathit{Int} \\
&\mathit{eval}\ (\mathit{Val}\ n) \quad\ = n \\
&\mathit{eval}\ (\mathit{Add}\ e\ e') = \mathit{eval}\ e + \mathit{eval}\ e'
\end{aligned}
$$

Que embedding elegir? (expression problem)

Shallow embedding

Pros: Es simple agregar nuevas construcciones al EDSL (por ejemplo, *mult*), mientras se puedan representar en el dominio de interpretación.

Cons: Agregar nuevas formas de interpretación (por ejemplo, hacer un pretty printing de las expresiones) puede implicar una reimplemantación completa.

Que embedding elegir? (expression problem)

Shallow embedding

Pros: Es simple agregar nuevas construcciones al EDSL (por ejemplo, *mult*), mientras se puedan representar en el dominio de interpretación.

Cons: Agregar nuevas formas de interpretación (por ejemplo, hacer un pretty printing de las expresiones) puede implicar una reimplemantación completa.

Deep embedding

Pros: Es simple agregar un nuevo observador (por ejemplo, pretty printing).

Cons: Agregar nuevas construcciones al lenguaje (como *mult*) implica modificar el tipo del AST (el tipo *Expr*) y reimplementar todos los observadores (las funciones de interpretación).

Razonamiento sobre el EDSL

A partir de la definición del EDSL en Haskell (tanto como shallow o deep embedding) es posible probar propiedades del EDSL.

Por ejemplo,

$$add\ e\ (add\ e'\ e'') \quad = \quad add\ (add\ e\ e')\ e''$$

$$add\ e\ e' \quad = \quad add\ e'\ e$$