MGL7460 — Automne 2016

Langages spécifiques au domaine

Guy Tremblay
Professeur

Département d'informatique
UQAM
http://www.labunix.uqam.ca/~tremblay

24 novembre 2016

Contenu

1. Introduction

Avez-vous déjà utilisé. . . make ?

```
$ cat Makefile
hello: hello.c
    gcc -o hello hello.c

clean:
    rm -f hello hello.o
```

```
$ make
gcc -o hello hello.c
```

Avez-vous déjà utilisé. . . SQL ?

```
SELECT title, price
  FROM  Book
   WHERE price > 30.00
  ORDER BY price;
```

```
Title                          Price
---------                      ---
DSLs in Action                 39.99
Domain-Specific Languages      45.99
Domain-Driven Design           49.99
```

Avez-vous déjà utilisé... HTML ?

```
<HTML>
<HEAD>
<TITLE>CSS Example</TITLE>
</HEAD>

<BODY>
<H1>Title of web page</H1>

<P>A first paragraph...</P>

<P>Another paragraph, with an
<A HREF=".">anchor
(reference)</A>...</P>
</BODY>

</HTML>
```

Si oui, alors vous avez déjà utilisé un DSL

= *Domain Specific Language*

= Langage Spécifique au Domaine

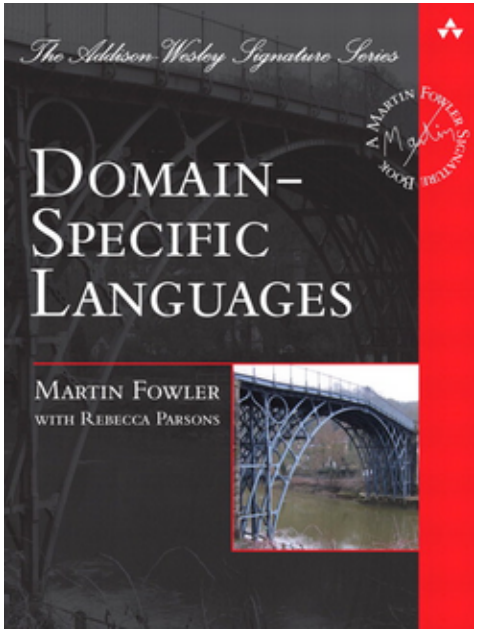
2. Qu’est-ce qu’un «Langage Spécifique au Domaine» ?

Qu'est-ce qu'un *Langage Spécifique au Domaine* (DSL) ?

Domain-specific language :

A computer programming language of limited expressiveness focused on a particular domain.

Source*: M. Fowler, 2011*



Qu'est-ce qu'un *Langage Spécifique au Domaine* (DSL) ?

Domain-specific language :

> *A language offering expressive power focused on a particular problem domain, such as a specific class of applications or application aspect.*

Source: *K. Czarnecki, 2005*

Qu'est-ce qu'un *Langage Spécifique au Domaine* (DSL) ?

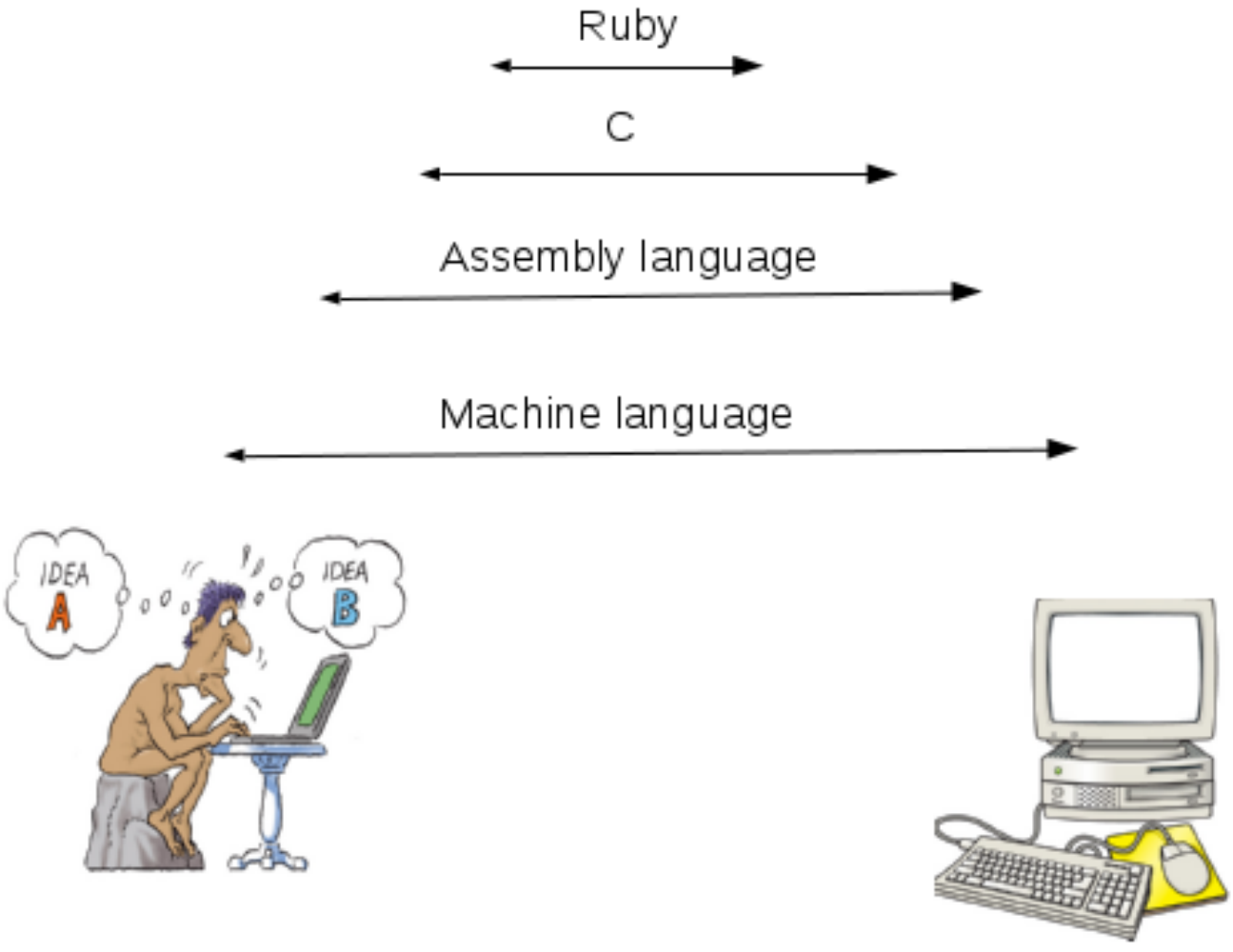
Domain-specific language :

> *A "little language", commonly known as a "domain-specific language" (DSL), is a language that is expressive uniquely over the specific features of programs in a given problem domain.*

Source: *A. Yepez, 2010*

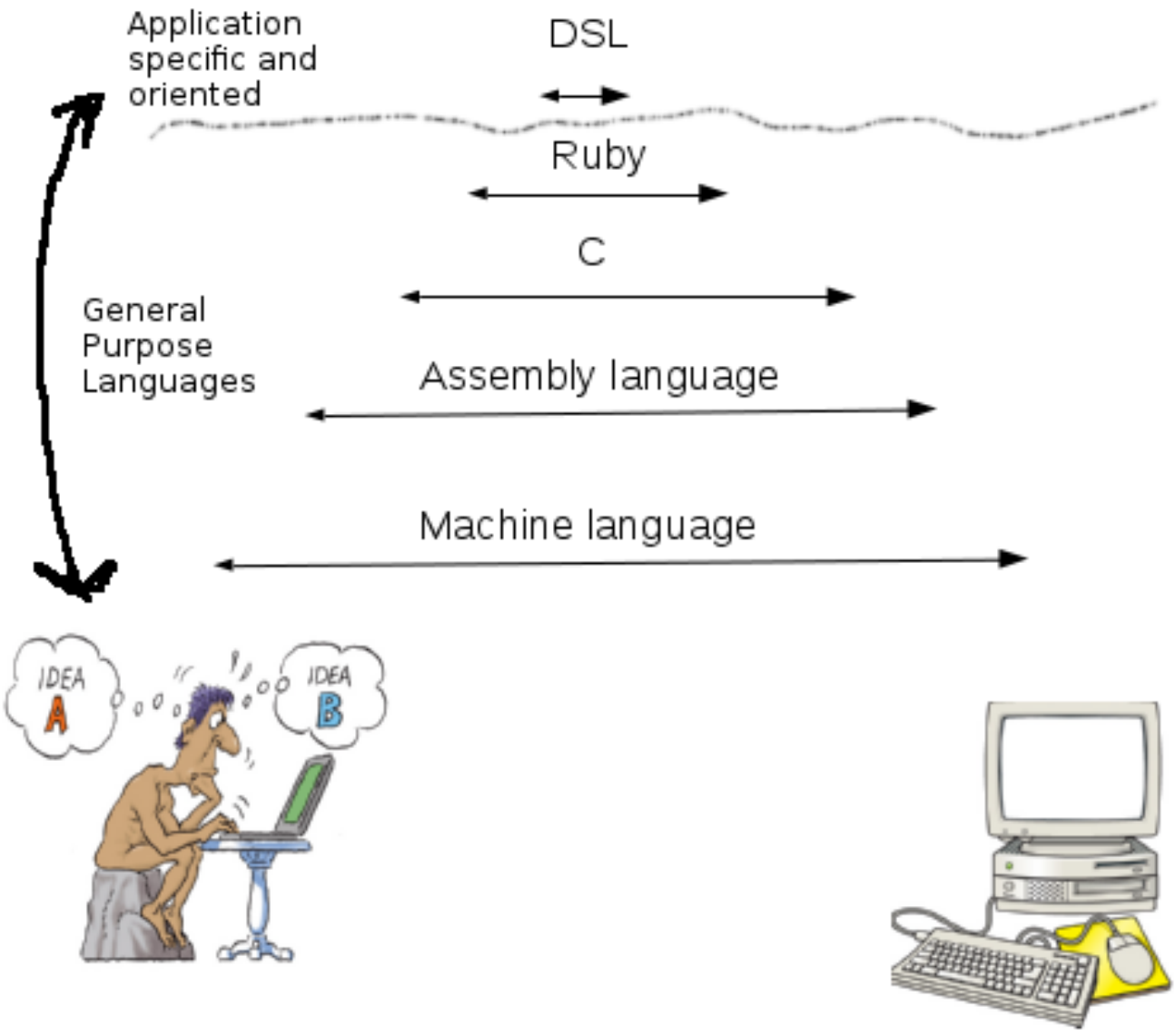
Un DSL vise à réduire le «fossé sémantique»

Le fossé entre idées de l'usager et leur expression dans un programme

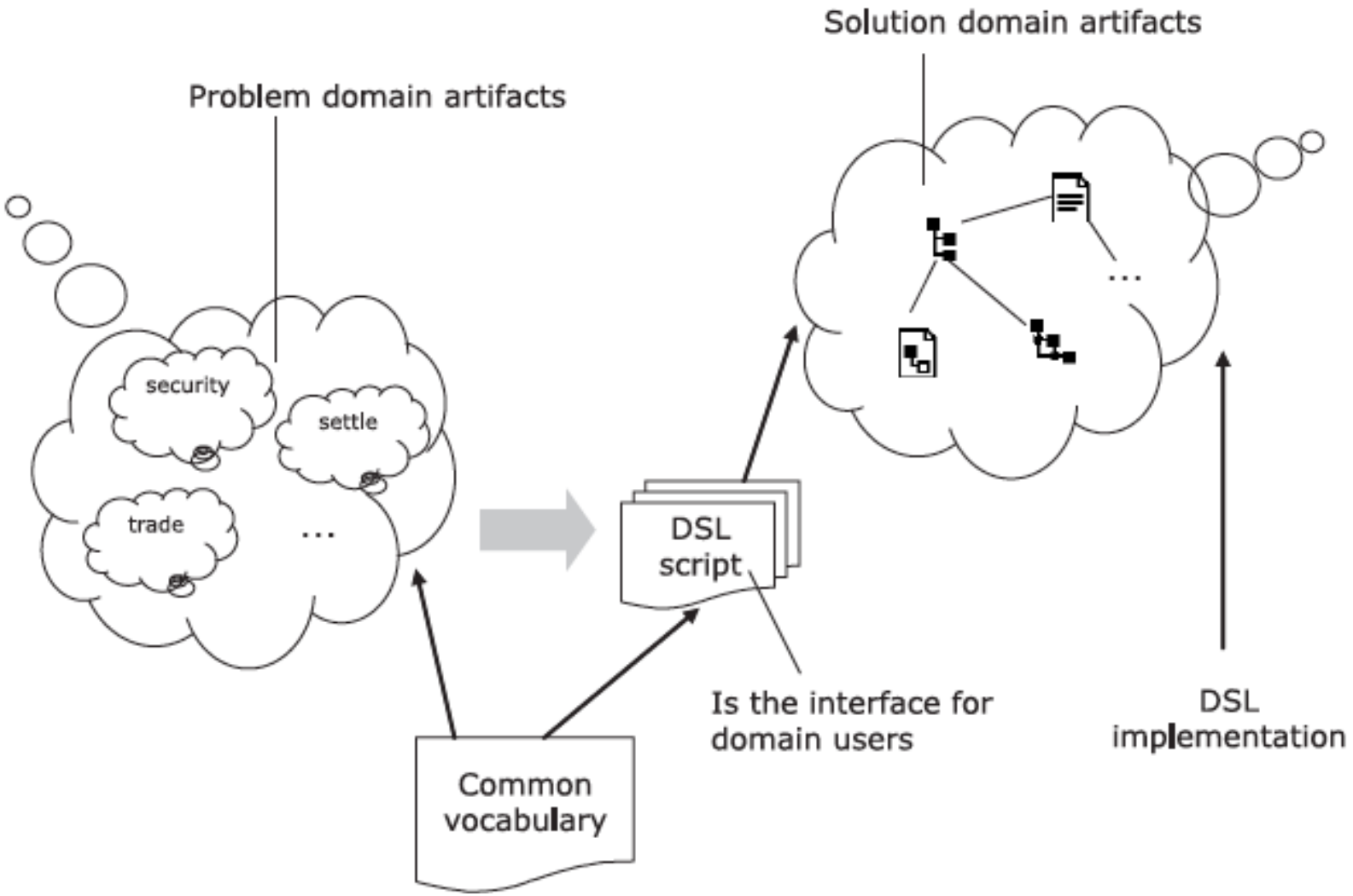


Un DSL vise à réduire le «fossé sémantique»

Le fossé entre idées de l'usager et leur expression dans un programme



Un DSL utilise le langage du domaine d'application



Source: "DSLs in Action", D. Ghosh, 2011

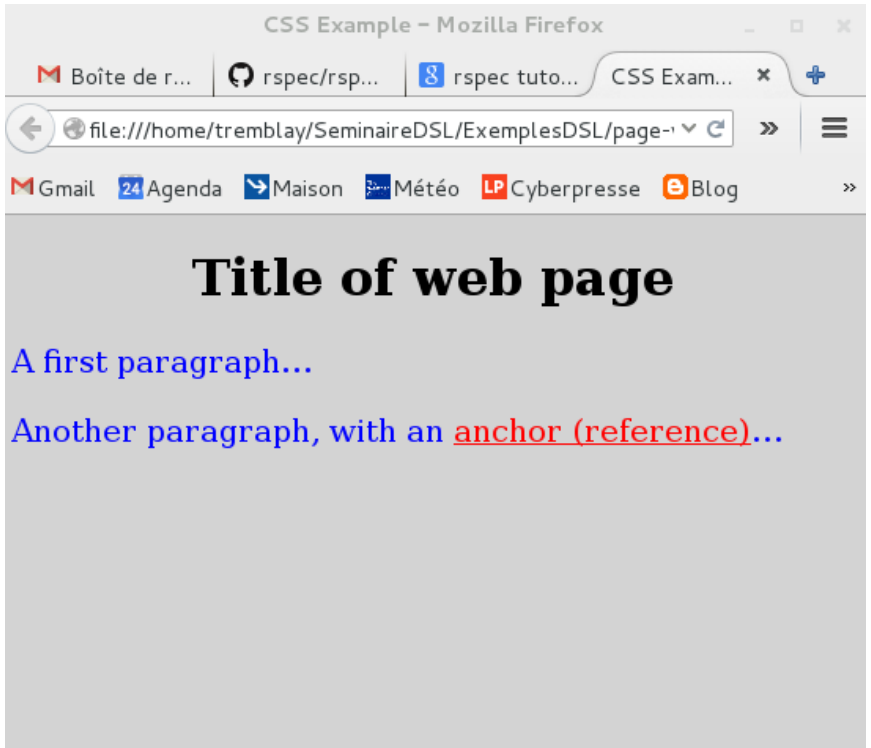
2.1 Quelques exemples de DSLs

HTML and CSS : HTML décrit la structure et le contenu

```
<!DOCTYPE HTML>
<HTML>
<HEAD>
<TITLE>CSS Example</TITLE>
</HEAD>



<BODY>
<H1>Title of web page</H1>

<P>A first paragraph...</P>

<P>Another paragraph, with an
<A HREF=".">anchor
(reference)</A>...</P>
</BODY>

</HTML>
```

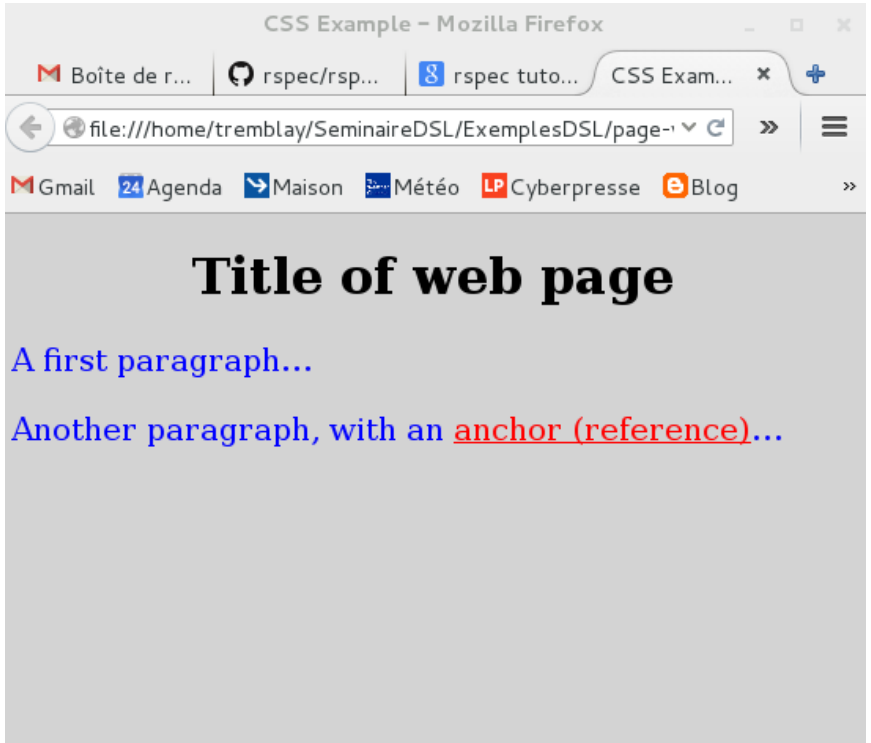
HTML and CSS : CSS décrit la mise en forme et le style

```
body {
background-color:lightgrey;
}

h1 {
  color: black;
  text-align: center;
}

p {
  font-size: 20px;
  color: blue;
}

a {
  color: red;
}
```



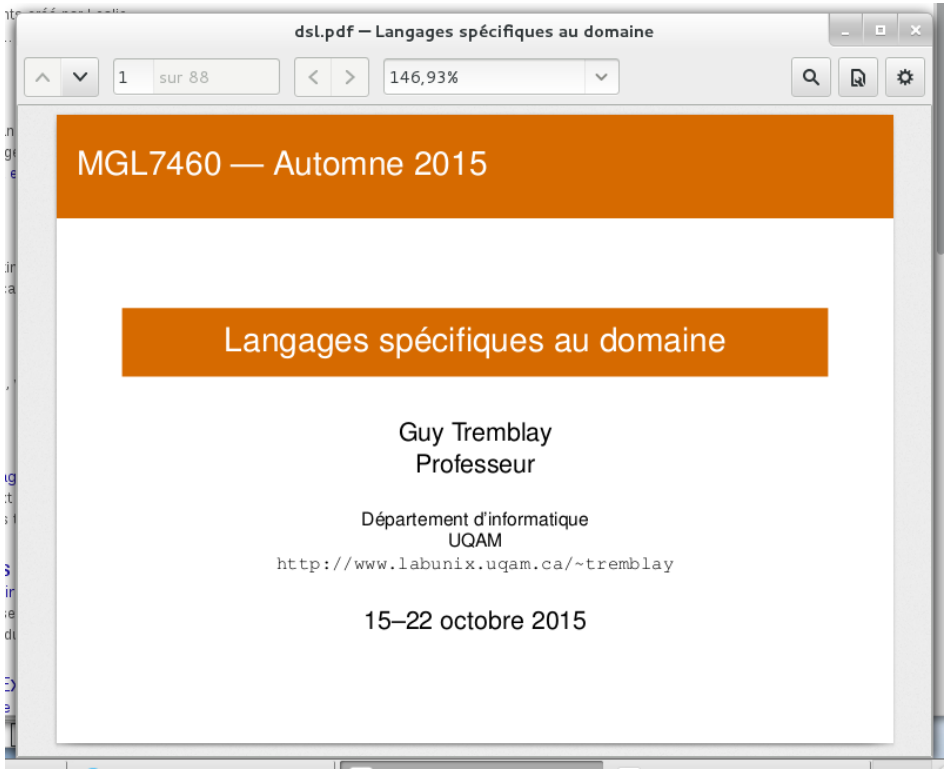
HTML and CSS : CSS décrit la mise en forme et le style

```
body {
background-color: lightgrey;
}

h1 {
  color: black;
  text-align: center;
}

p {
  font-size: 20px;
  color: blue;
}

a {
  color: red;
}
```



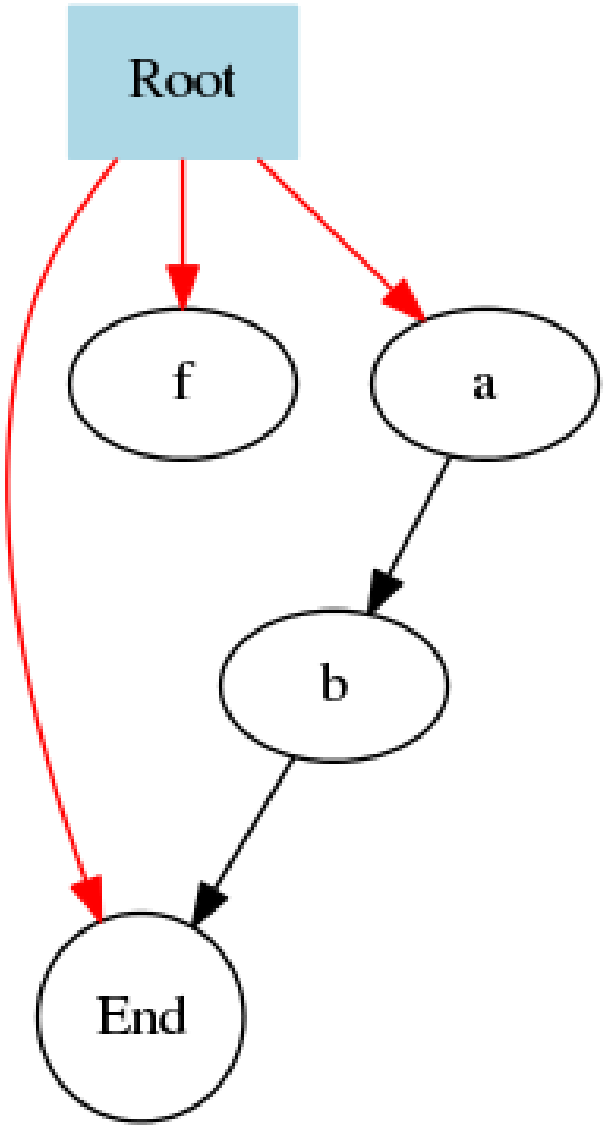
LaTeX : Composition de documents

```
$ cat dsl.tex

\documentclass[t]{beamer}
\usepackage[french]{babel}
\usepackage[utf8]{inputenc}
...

\begin{document}

\begin{frame}
\frametitle{MGL7460...}
\titlepage
\end{frame}
...

\end{document}

$ pdflatex dsl; evince dsl.pdf
```



graphviz : Spécification de graphes

```
$ cat G1.dot
digraph G1 {

  Root [shape=box,
        color=lightblue,
        style=filled]
  End  [shape=circle]

Root -> f; a; End [color=red]
   a -> b -> End;
}

$ dot -Tpng G1.dot  > G1.png
```



Make : Outil Unix d'assemblage de logiciels

Utilise une syntaxe simple et *ad hoc* — tabs ≠ espaces

```
$ cat hello.c
#include <stdio.h>

int main() {
  printf( "Hello, World!\n" );
}

$ cat Makefile
hello: hello.c
        gcc -o hello hello.c

clean:
        rm -f hello hello.o
```

```
$ make
gcc -o hello hello.c

$ ./hello
Hello, World!

$ make clean
rm -f hello hello.o
```

Source: `http://hyperpolyglot.org/build`

Ant : Outil Java d'assemblage de logiciels

Utilise une description XML des dépendances et des règles

```
$ cat Hello.java
public class Hello {
  public static void main() {
    System.out.println(
      "Hello, World!" );
  }
}
```

```
$ cat build.xml
<project default="hello">
  <target name="hello">
    <javac srcdir="."
       includeantruntime="false
       destdir="."/>
  </target>

  <target name="clean">
    <delete>
      <fileset dir="."
         includes="*.class"/>
    </delete>
  </target>
</project>
```

```
$ ant
Buildfile: build.xml

hello:
    [javac] Compiling 1 source
            file to [...]

BUILD SUCCESSFUL
Total time: 3 seconds

$ java Hello
Hello, World!

$ ant clean
Buildfile: build.xml

clean:

BUILD SUCCESSFUL
Total time: 0 seconds
```

Rake : Outil Ruby d'assemblage de logiciels

Utilise du Ruby standard ⇒ DSL interne

```
$ cat hello.c
#include <stdio.h>

int main() {
  printf( "Hello, World!\n" );
}

$ cat Rakefile
task :default => "hello"

file "hello" => ["hello.c"] do
  sh "gcc -o hello hello.c"
end

task :clean do
     rm_f "hello hello.o"
end
```

```
$ rake
gcc -o hello hello.c

$ ./hello
Hello, World!

$ rake clean
rm -f hello hello.o
```

Source: `http://hyperpolyglot.org/build`

`gli` : Spécification d'interface en ligne de commandes

Utilises du Ruby standard ⇒ DSL interne

- `gli` = *g*`it` *l*`ike` *i*`nterface` *command line parser*

= DSL pour définir *des suites de lignes de commandes*

`gli` : Spécification d'interface en ligne de commandes

Une interface en ligne de commandes pour prêter et rapporter des livres

```
$ bin/biblio emprunter "Guy T." tremblay.guy@uqam.c
        "The RSpec Book" "Chelimsky et al."


$ bin/biblio emprunteur "The RSpec Book"
 Guy T.


$ bin/biblio rappeler_livre "The RSpec Book"
 Un courriel a ete envoye a tremblay.guy@uqam.ca.


$ bin/biblio rapporter "The RSpec Book"
```

`gli` : Spécification d'interface en ligne de commandes

Mise en oeuvre de `bin/biblio` avec `gli`

```
desc "Indique l'emprunt d'un livre (ou [..] stdin)"
arg_name "nom courriel titre auteurs"
command :emprunter do |c|
  c.action do |global_options,options,args|
    verifier_nb_args args, 4

    avec_biblio( global_options[:depot] ) do |bib|
      bib.emprunter( *args )
    end
  end
end
```

`gli` : Spécification d'interface en ligne de commandes

Mise en oeuvre de `bin/biblio` avec `gli` (suite)

```
desc "Indique le retour d'un livre"
arg_name 'titre'
command :rapporter do |c|
  c.action do |global_options,options,args|
    verifier_nb_args args, 1
    titre = args[0]

    avec_biblio( global_options[:depot] ) do |bib|
      bib.rapporter( titre )
    end
  end
end
```

`gli` : Spécification d'interface en ligne de commandes

Mise en oeuvre prédéfinie pour «`bin/biblio help`»

```
<<~/ExemplesCucumber/Biblio@MacBook>> $ biblio help
NAME
    biblio - Programme pour la gestion de prets de livres

SYNOPSIS
    biblio [global options] command [command options] [arguments...]

VERSION
    0.4.0

GLOBAL OPTIONS
    --depot=depot - Fichier contenant le depot (default: ./.biblio.txt)
    --help        - Show this message
    --version     - Display the program version

COMMANDS
    emprunter               - Indique l'emprunt d'un livre (ou de plusieurs via stdin)
    emprunteur              - Determine l'emprunteur d'un livre
    emprunts                - Retourne les livres empruntes par quelqu'un
    help                    - Shows a list of commands or help for one command
    indiquer_perte          - Indiquer la perte d'un livre
    init                    - Cree une nouvelle base de donnees pour gerer des livres emprunte
                              './.biblio.txt' si --depot n'est pas specifie)
    lister                  - Liste l'ensemble des livres empruntes
    rappeler_livre          - Transmet un courriel a l'emprunteur d'un livre pour lui demander
                              rapporter
    rappeler_tous_les_livres - Transmet des courriels a tous les emprunteurs pour leur demander
                              rapporter
    rapporter               - Indique le retour d'un livre
    trouver                 - Retourne le titre complet d'un livre (ou tous les titres qui con
                              la chaine)
```

RSpec : Un cadre de test de style BDD

Utilise du Ruby standard ⇒ DSL interne

```
describe Hash do
  before(:each) do
    @ages = {::Thomas => 7, Nellie => 10}
  end

  describe "#[]" do
    it "adds a key" do
      @ages[:Laurent] = 5
      @ages.should == {:Laurent => 5,
                       :Thomas  => 7, :Nellie => 10}
    end

    it "replaces an existing key" do
      @ages[:Nellie] = 11
      @ages.should == {:Thomas => 7, :Nellie => 11}
    end
  end
```

RSpec : Un cadre de test de style BDD

Utilise du Ruby standard ⇒ DSL interne

```
describe "#clear" do
  it "deletes all keys" do
    @ages.clear
    @ages.keys.should be_empty
  end
end

describe "#merge!" do
  it "merges elements from another hash" do
    @ages.merge!( {:Laurent => 5} )
    @ages.should have_exactly(3).items
  end
end

describe "#fetch" do
  it "raises an error when key not found" do
    ->{ @ages.fetch(:Michelle) }.should raise_error(KeyError)
  end
end
```

RSpec : Un cadre de test de style BDD

Utilise du Ruby standard ⇒ DSL interne

```
$ rspec -I. . --format=documentation

Hash
  .new
    creates an empty hash
  #[]
    adds a key
    replaces an existing key
  #delete
    deletes a key
  #clear
    deletes all keys
  #merge!
    merges elements from another hash
  #fetch
    raises an error when key not found

Finished in 0.0062 seconds (files took 0.08588 seconds to load
7 examples, 0 failures
```

Financial Brokerage : Vente et achat d'*actions*

Utilise du Scala standard $\Rightarrow$ DSL interne

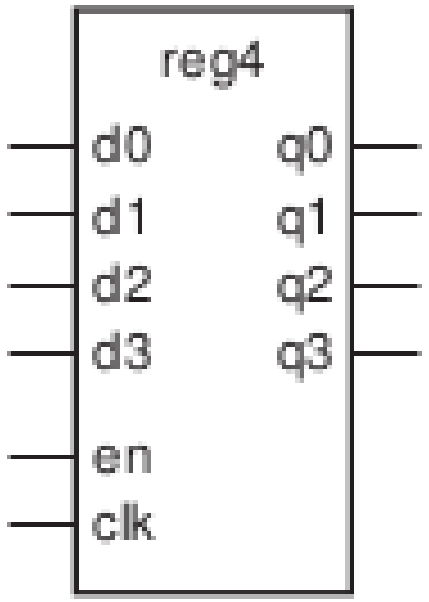
```
orders = []

ord1 = newOrder.to.buy( 100.shares.of('IBM') {
  limitPrice 300
  allOrNone true
  valueAs { qty, unitPrice -> qty * unitPrice - 500 }
}
orders << ord1

ord2 = newOrder.to.buy( 200.shares.of('GOOG') {
  limitPrice 500
  allOrNone true
  principal = 10000
  valueAs { qty, unitPrice -> qty * unitPrice - 500 }
}
orders << ord2
```

Source: "DSLs in Action", D. Ghosh, 2011

VHDL : Description de composants matériels

VHDL = *VHSIC Hardware Description Language*

VHSIC = *Very High Speed Integrated Circuits*



Source: `http://ati.ttu.ee/~alsu/VHDL_TUTORIAL_Ashenden.pdf`

VHDL : Description de composants matériels

Description de l'interface

```
entity reg4 is
        port ( d0, d1, d2, d3, en, clk : in bit;
                q0, q1, q2, q3 : out  bit );
end entity reg4;
```

Source: `http://ati.ttu.ee/~alsu/VHDL_TUTORIAL_Ashenden.pdf`

VHDL : Description de composants matériels

Description du comportement

```
architecture behav of reg4 is
begin
    storage : process is
        variable stored_d0, stored_d1, stored_d2, stored_d3 : bit;
    begin
        wait until clk = '1';
        if en = '1' then
            stored_d0 := d0;
            stored_d1 := d1;
            stored_d2 := d2;
            stored_d3 := d3;
        end if;
        q0 <= stored_d0 after 5 ns;
        q1 <= stored_d1 after 5 ns;
        q2 <= stored_d2 after 5 ns;
        q3 <= stored_d3 after 5 ns;
    end process storage;
end architecture behav;
```

Source: `http://ati.ttu.ee/~alsu/VHDL_TUTORIAL_Ashenden.pdf`

CVME : Un *gem* pour écrire des *Curriculum Vitae*

Source: `https://github.com/Yorgg/cvme`

```
Cvme.create('/Users/user/desktop/cv.html', 'default') do
  header do
      user    'Jean Luc Picard'
      email   'jean@gmail.com'
      ...
   end

  group 'Work experience' do
    entry 'Captain - USS Enterprise-D' do
      date '2364-'
      description 'Captain of the latest galaxy starship.'
      b1 'Evaded mutiple Borg attacks'
      ...
    end
  end

  ...
end
```

CVME : Un *gem* pour écrire des *Curriculum Vitae*

Jean Luc Picard

12 Rue de montagne
La Barre, Franche-Comté, France
1286493820
jean@gmail.com

Work experience

Captain - USS Enterprise-D 2364-

Captain of the latest galaxy class starship.

- Evaded multiple Borg attacks
- Succesfully held the the Romulan-Klingon border
- Managed relations with Q
- Lived on the planet Kataan for 40 years

Lieutenant Commander/Captain - USS Stargazer 2333-2356

Commanding officer on the USS Stargazer

- Defended the Stargazer from over 12 Cardassian attacks
- Managed relations with Cardassians, Ferengi, Klingons, and Vulcans

Projects

Simulated Universe 2353

Helped in the creation of a simulated universe for Dr. Moriarty and his wife

Archeology: Uncovered a 21st century DSL written in Ruby 2355

view

Interests

- Fencing
- Racquetball
- Equine sports
- Archeology

Oto : Un programme pour aider à la correction des travaux de programmation

Utilise du Ruby standard ⇒ DSL interne

```
$ cat script-correction.oto
groupe.each do |tp|
  comp = compiler_javac( tp ) {
    :fichier >> "Tp1.java"
  }
  tests = tester_junit( tp ) {
    :classe >> "TestTp1"
  }
  nbErreurs = tests[:nberreurs]
  nbTests = tests[:nbtests]

  tp["Nb. tests"]    = nbTests
  tp["Nb. erreurs"] = nbErreurs
  tp["Note finale"] = 100.0 * ( nbTests - nbErreurs ) / nbTests
end

puts rapport_complet( groupe )
```

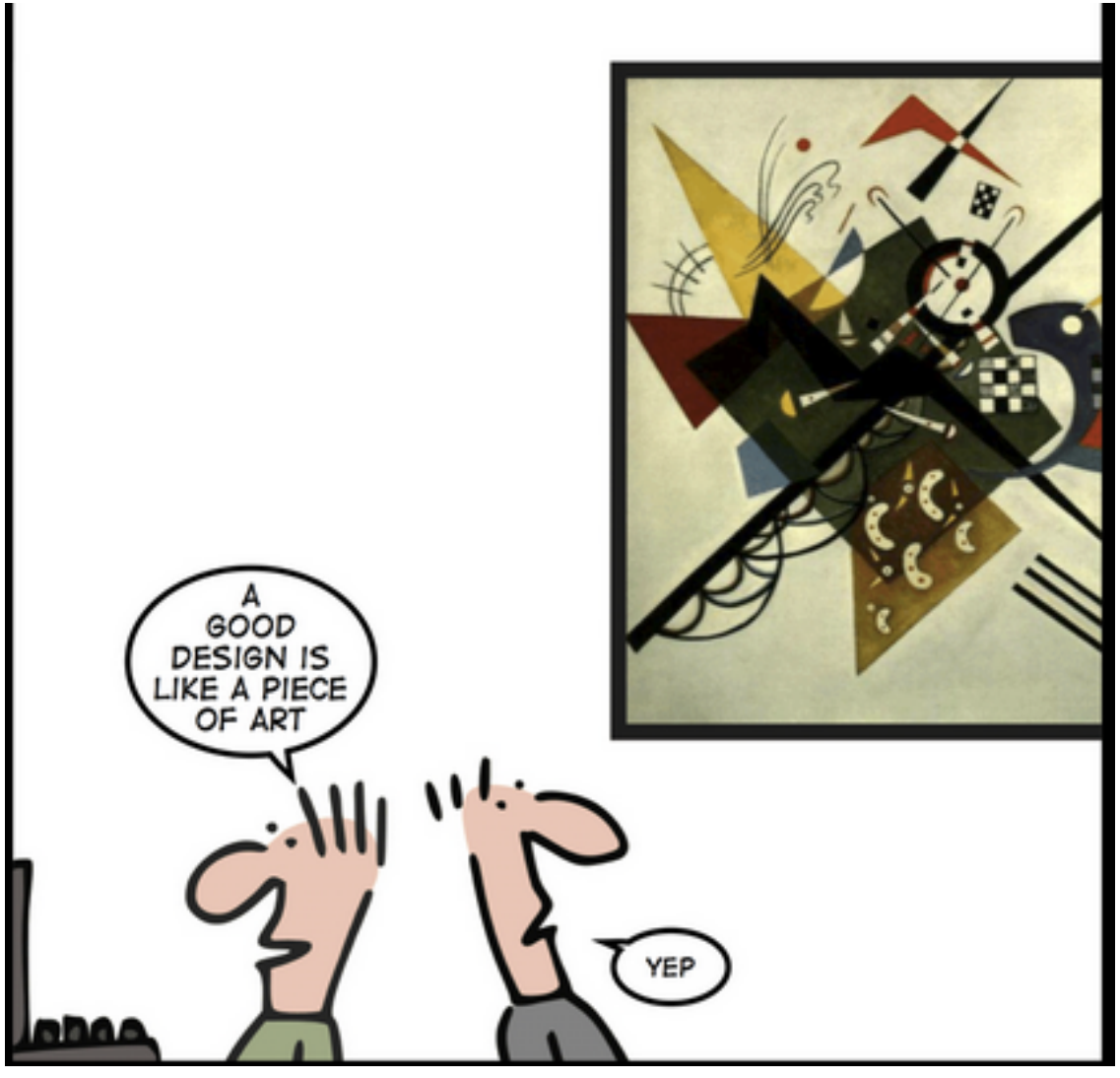
Oto : Un programme pour aider à la correction des travaux de programmation

Utilise du Ruby standard ⇒ DSL interne

```
$ oto script-correction.oto TravauxRemis/*.tp_oto
...

TRAVAIL: tremblay+2010.06.18.08.41.917949+TREG05065801.tp_oto
Equipe:         DURN27018400
Depot:          2010-06-18 at 08:41
Deposeur:       tremblay
Nom:            tremblay
Courriel:       ???

RESULTATS:
  Nb. tests:
      4
  Nb. erreurs:
      0
  Note finale:
      100.0

...
```

2.2 Pourquoi utiliser des DSLs ?

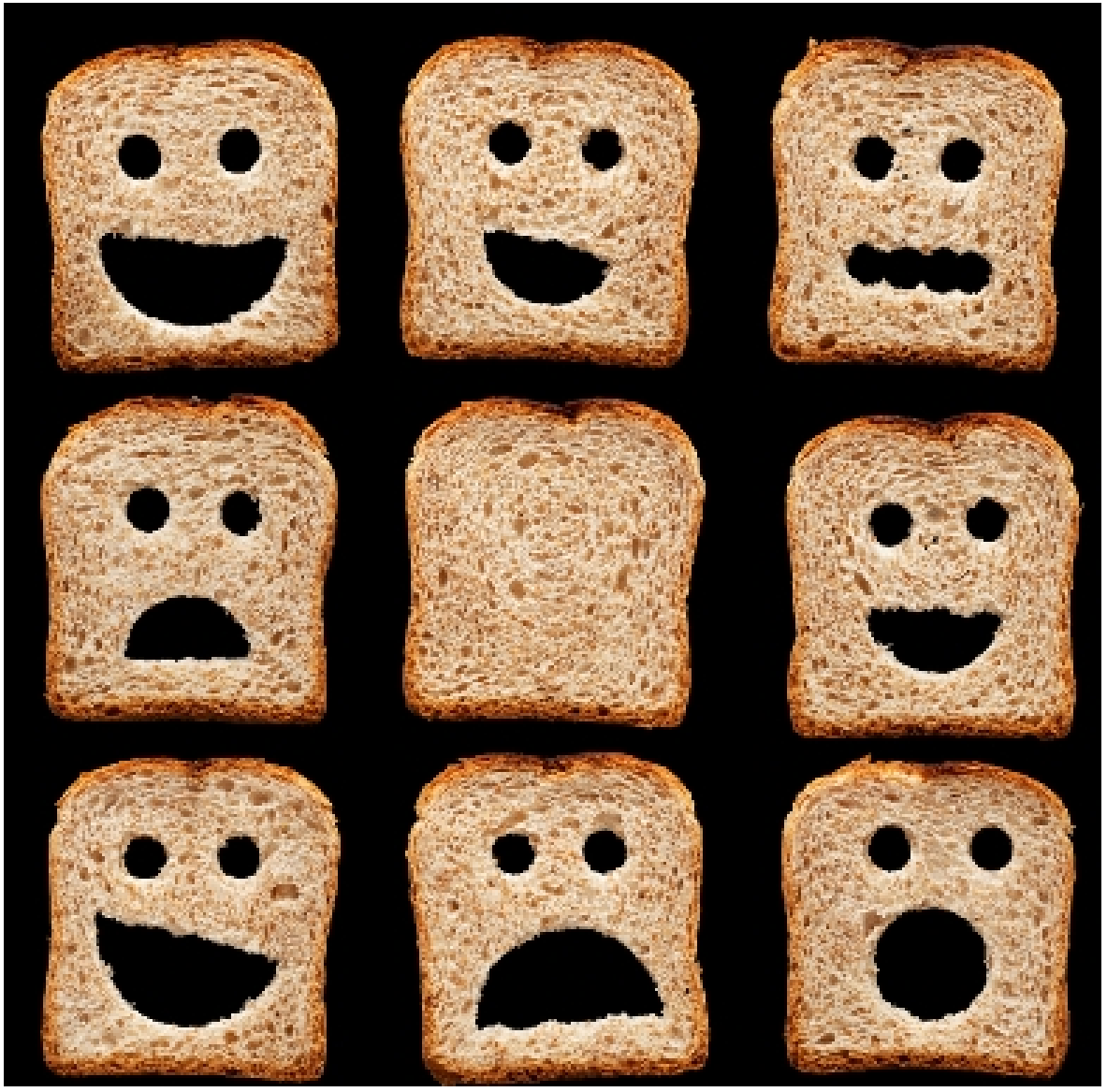
Un DSL permet un meilleur niveau d'abstraction



Source:

`geek-and-poke.com`

Un DSL est un langage plus expressif



Un DSL facilite la comunication avec les experts du domaine

"I know nothing about the subject,
but I'm happy to give you my expert opinion."

Désavantages des DSLs

- Concevoir un langage est difficile :
  DSL $\Rightarrow$ Connaissance du Domaine

- Un autre langage (nouveau) à apprendre

- Une autre couche logicielle

2.3 Différentes sortes de DSLs

DSL externe vs. DSL interne : Caractérisation de Fowler

DSL externe

"An external DSL is a completely separate language, for which you [need] a full parser."

Source: *Fowler, 2009*

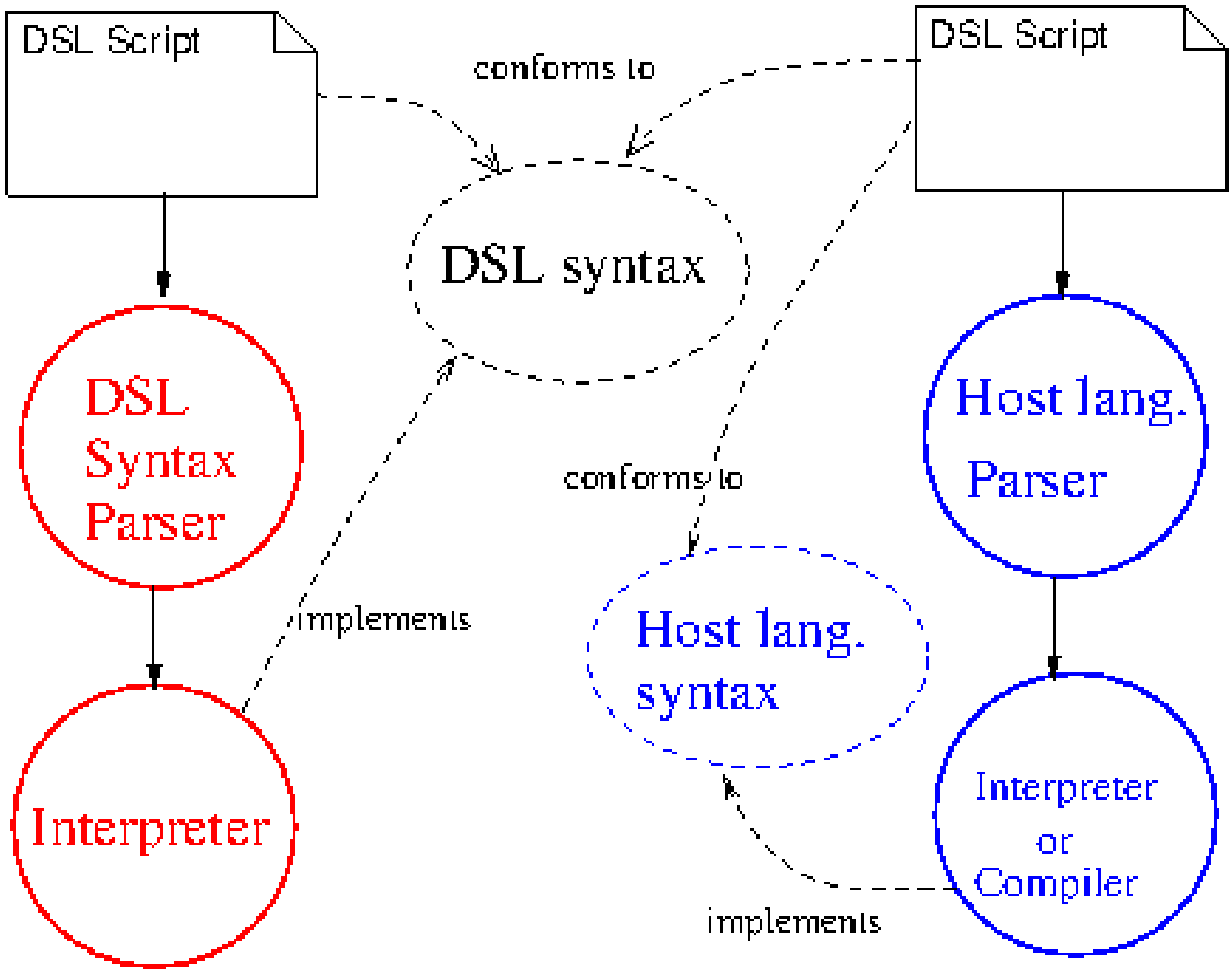
DSL interne

An internal DSL is an idiomatic way of using a general-purpose language."

Source: *Fowler, 2009*

External DSL

Internal DSL



DSL externe vs. DSL interne : Exemples

DSL Externes

- HTML
- CSS
- Graphviz
- Make
- Ant

DSL Internes

- Rake
- Gli
- RSpec
- Scripts de correction Oto

DSL autonome vs. DSL fragmentaire : Caractérisation de Fowler

DSL autonome

"[A stand-alone DSL is used through a] DSL script, typically in a single file, and it is all DSL."

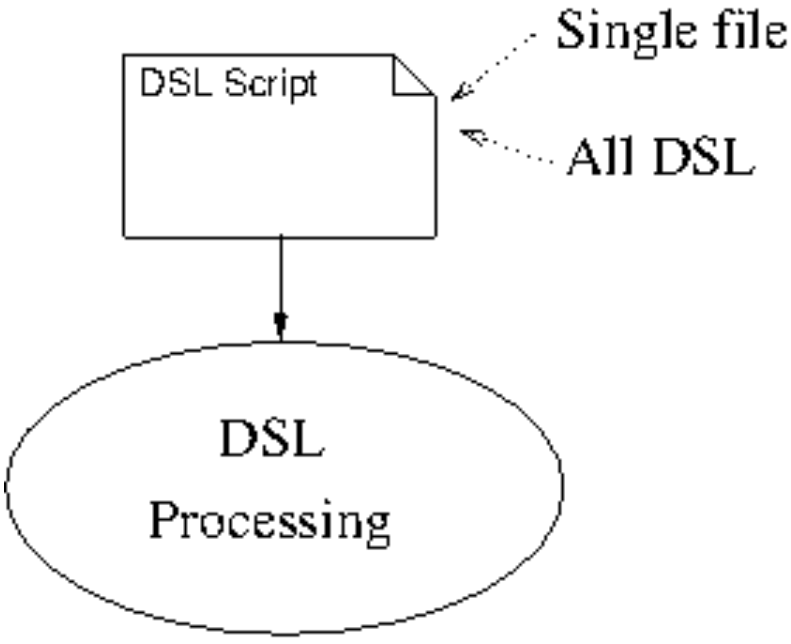
Source: *Fowler, 2009*

DSL fragmentaire

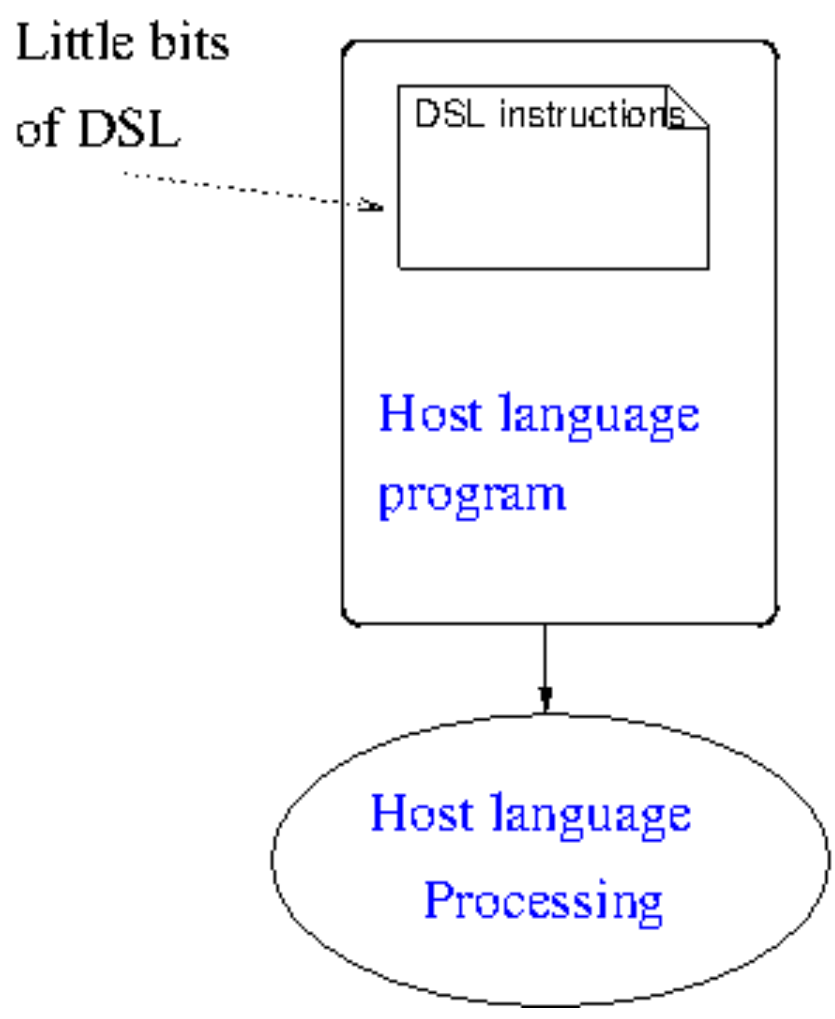
"[With a fragmentary DSL,] little bits of DSL are used inside the host language code. You can think of them as enhancing the host language with additionnal features."

Source: *Fowler, 2009*

Stand–alone DSL



Fragmentary DSL



DSL autonome vs. DSL fragmentaire : Exemples

DSL Autonomes

- HTML
- CSS
- graphviz
- Make
- Ant
- Rake
- Scripts de correction Oto

DSL Fragmentaires

- Gli
- Expressions régulières
- SQL enchassé

DSL applicatif vs. DSL utilitaire : Caractérisation de Voelter

Application domain DSL

[These] DSLs describe the core business logic of an application system independent of its technical implementation. These DSLs are intended to be used by domain experts, usually non-programmers.

Utility DSL

[These] DSLs [act] simply as utilities for developers. A developer, or a small team of developers, creates a small DSL that automates a specific, usually well-bounded aspect of software development.

Source: «DSL Engineering : Designing, Implementing and Using Domain-Specific Languages», Voelter

Les différentes catégories : Exemples

	Externes	Internes
Autonomes	HTML CSS Make Ant Graphviz LaTeX ...	Rake Scripts Oto
Fragmentaires	SQL enchassé	Gli RSpec

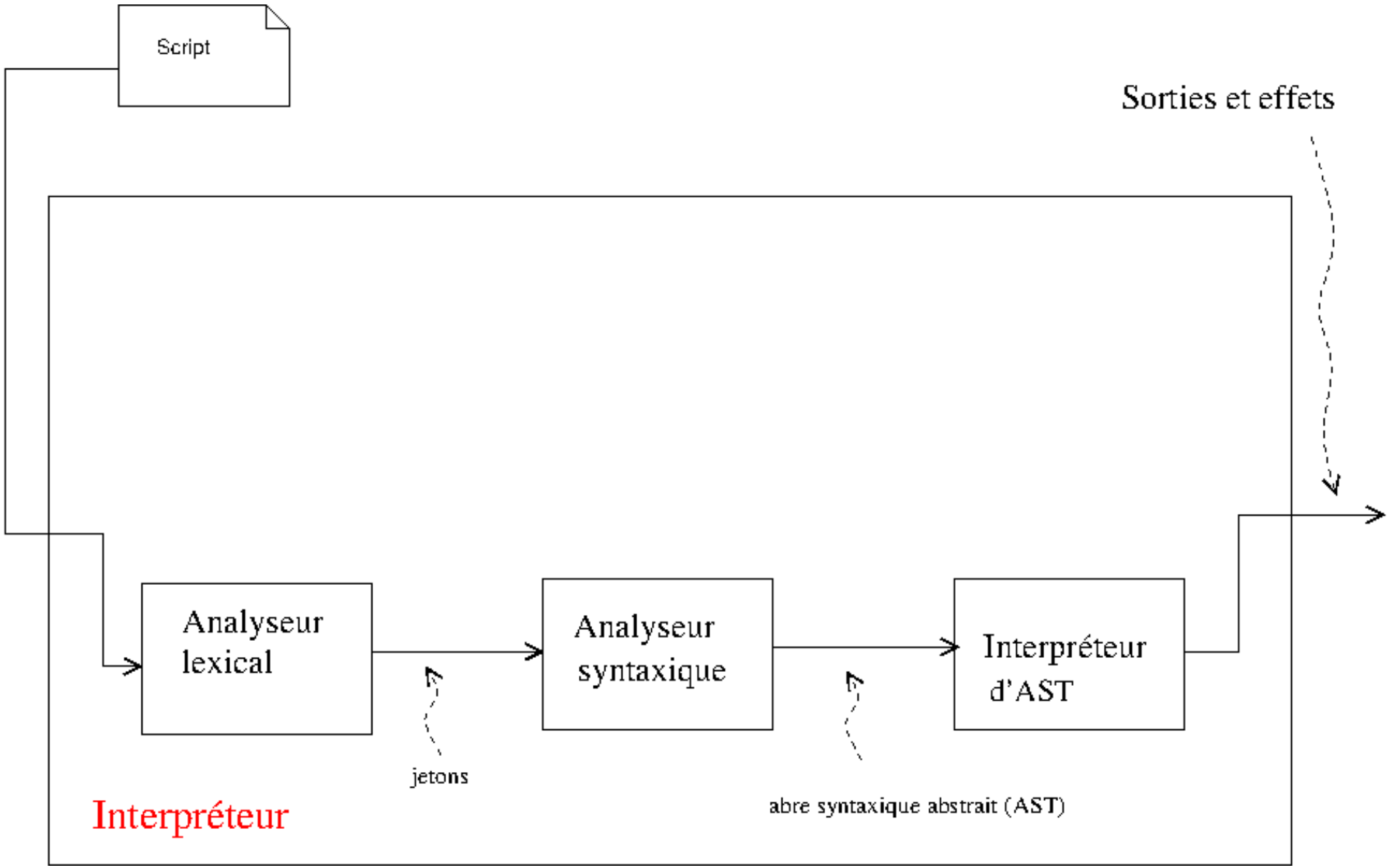
Légende : Utilitaire

3. Mise en oeuvre des DSLs

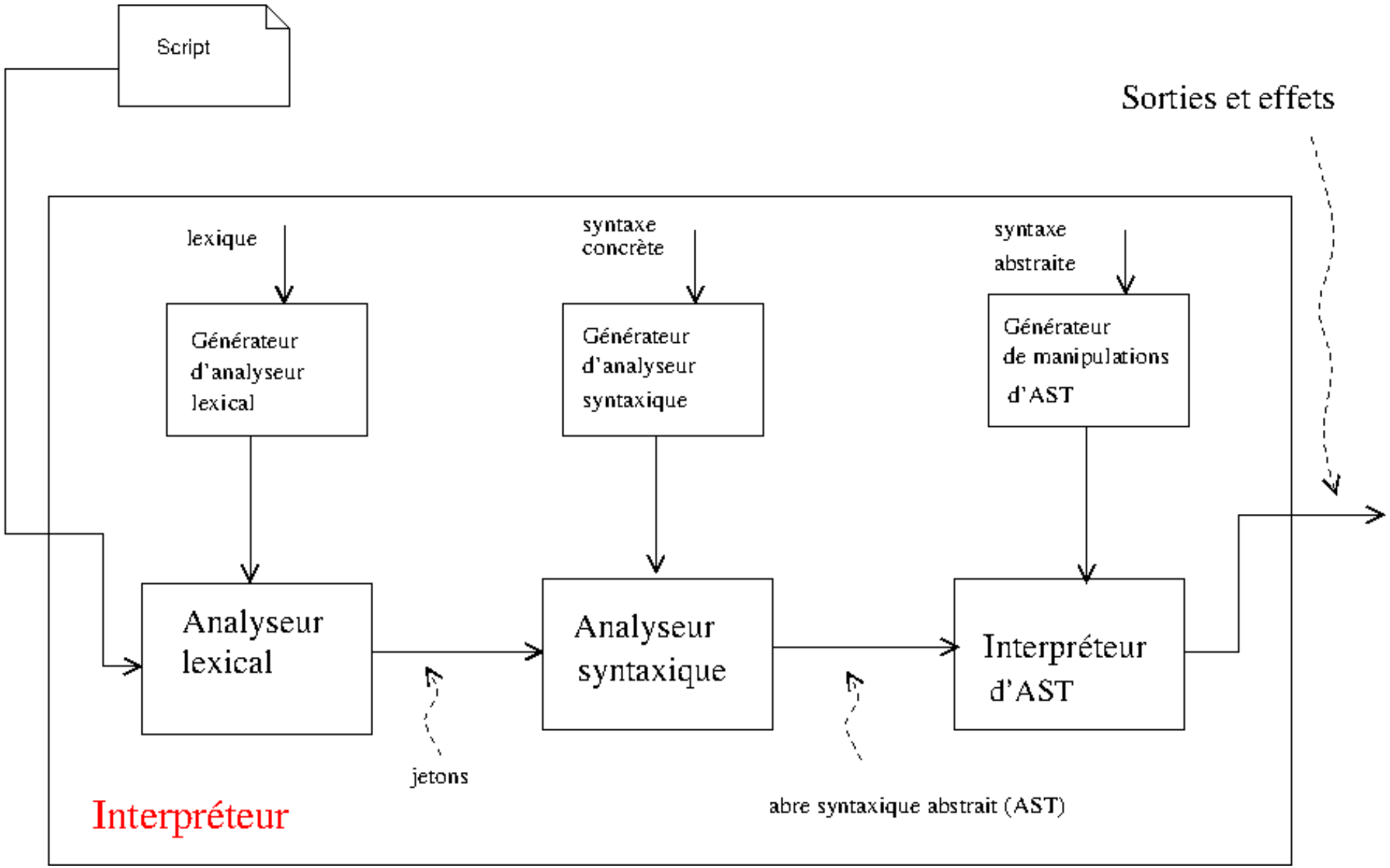
3.1 Mise en oeuvre des DSLs externes

Mise en oeuvre «classique» d'un DSL externe

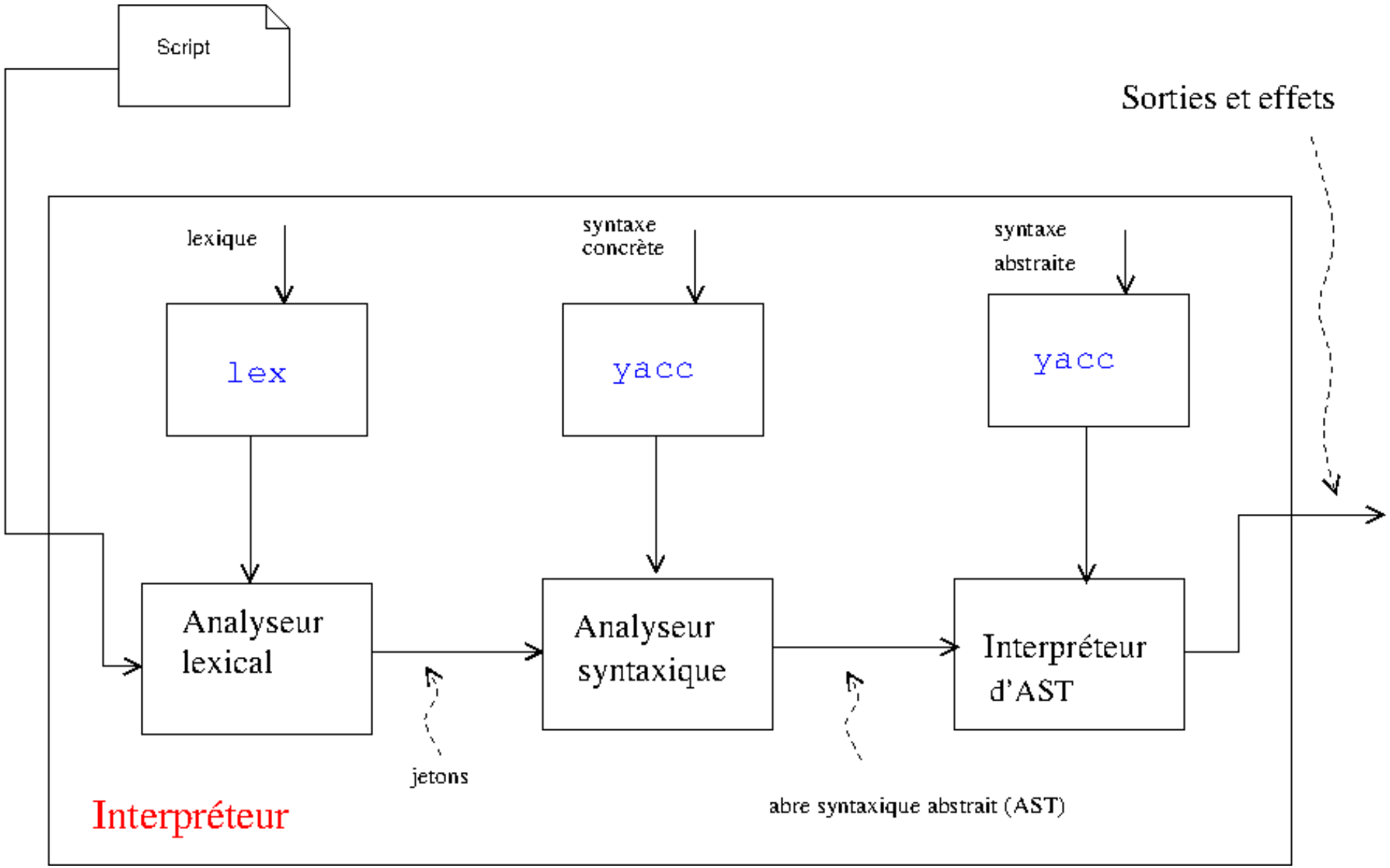
Mise en oeuvre «classique» d'un DSL externe



Mise en oeuvre «classique» d'un DSL externe



Mise en oeuvre «classique» d'un DSL externe



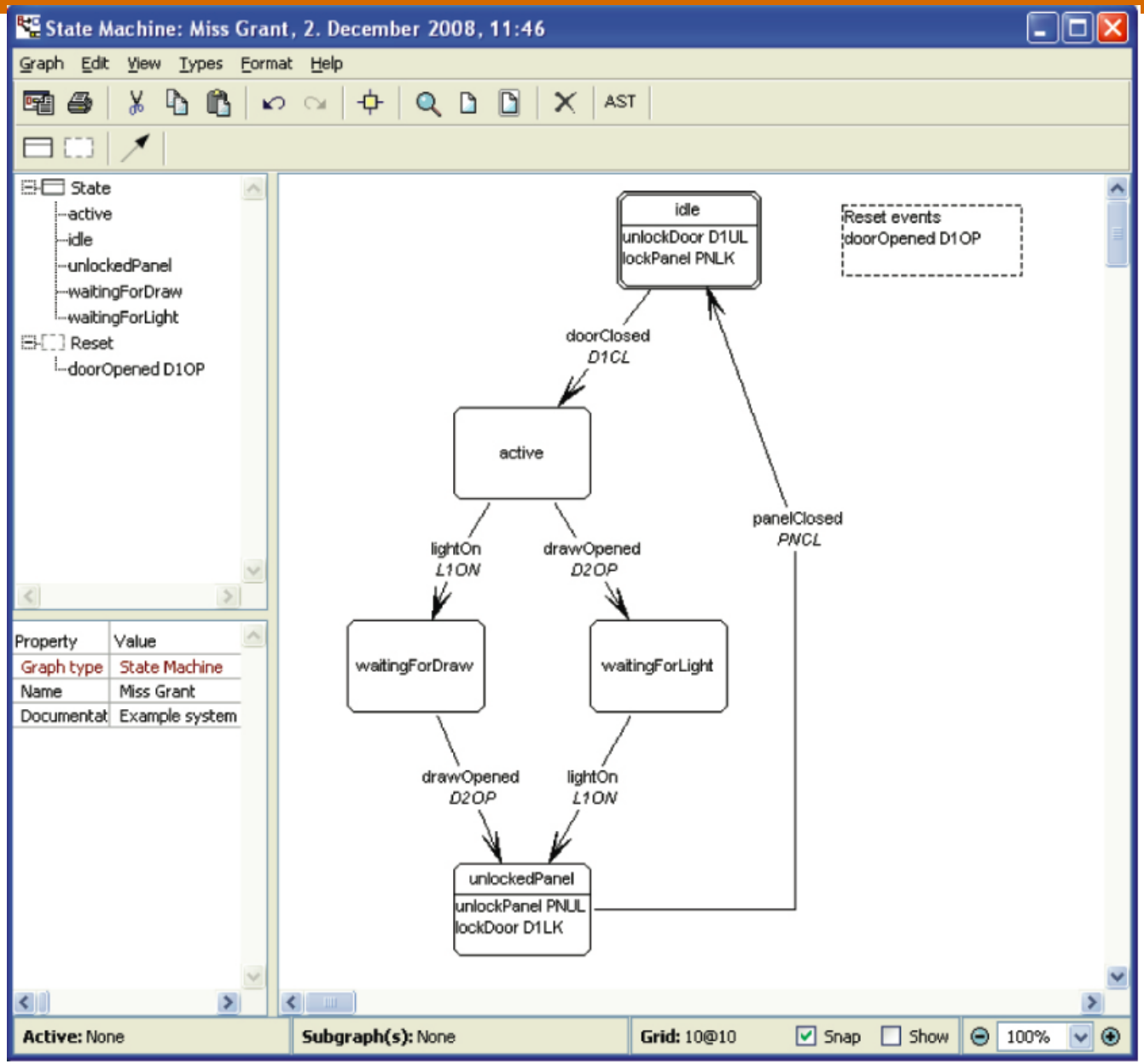
Plus récemment : *language workbenches*

A Language Workbench (LWB) is a development toolset that facilitates the development and editing of domain specific languages (DSLs).

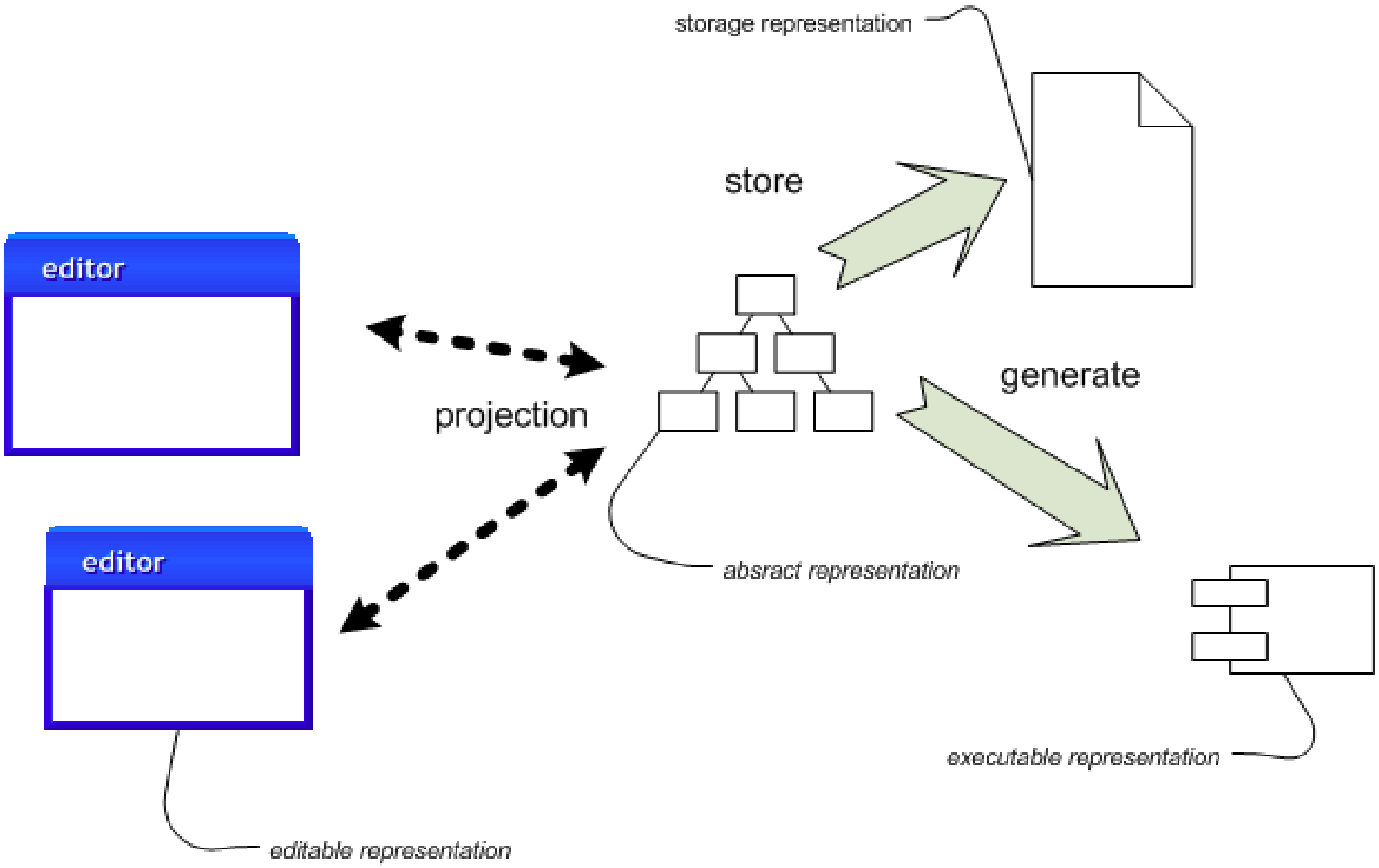
Source*:*

http://searchsoftwarequality.techtarget.com/definition/Language-Workbench

Plus récemment : *language workbenches*



Plus récemment : *language workbenches*



3.2 Mise en oeuvre des DSLs internes

Un DSL interne peut être vu comme une «API bien conçue», comme une interface coulante

API Régulière vs. DSL Interne

"[By contrast with a regular API], an internal DSL should have the feel of putting together whole sentences, rather than a sequence of disconnected commands."

Source: *Fowler, 2011*

Un DSL interne peut être vu comme une «API bien conçue», comme une interface coulante

Un DSL devrait définir une interface applicative «*coulante*»

"A **fluent interface** *is a way of implementing an object oriented API in a way that aims to provide for more readable code."*

Source*: Evans & Fowler, 2005*

Des techniques de mises en oeuvre de DSLs internes

Voir sections suivantes. . .

- Avec des APIs régulières
- Avec des APIs coulantes

4. Divers DSLs internes pour décrire des documents

4.1 Le problème : Décrire des documents avec attributs variés

Les DSLs sont souvent utilisés pour la spécification ou la configuration de systèmes ou d'objets complexes

- Dans ce qui suit, on va spécifier/décrire des `Documents`.

- Ces objets — pour alléger la présentation — seront relativement simples : quatre (4) attributs seulement

Les DSLs sont souvent utilisés pour la spécification ou la configuration de systèmes ou d'objets complexes

- Mais, pour motiver la discussion qui suit, imaginez qu'on ait plutôt des dizaines de propriétés — par exemple, description et spécification d'un *gem* :

```
spec = Gem::Specification.new do |s|
  s.name = 'mini-sed'
  s.version = MiniSed::VERSION
  s.author = 'Your Name Here'
  s.email = 'your@email.address.com'
  s.homepage = 'http://your.website.com'
  s.platform = Gem::Platform::RUBY
  s.summary = 'Une version simplifiee de sed'
  s.description = 'Une version simplifiee de sed, avec des commandes a la git. Utilise comme l
  s.files = ...
  s.require_paths << 'lib'
  s.has_rdoc = true
  s.extra_rdoc_files = ['README.rdoc','mini-sed.rdoc']
  s.rdoc_options << '--title' << 'mini-sed' << '--main' << 'README.rdoc' << '-ri'
  s.bindir = 'bin'
  s.executables << 'mini-sed'
  s.add_development_dependency('rake', '~> 11.1')
  s.add_development_dependency('rdoc', '~> 0')
  s.add_development_dependency('aruba', '0.8.1')
  s.add_development_dependency('minitest', '5.4.3')
  s.add_runtime_dependency('gli','2.12.0')
end
```

Les caractéristiques d'un `Document`

Attributs obligatoires

- `titre`
- `auteurs`
- `annee` (de publication, donc un entier)

Attribut optionnel

- `editeur`

Contrainte de création d'un objet (invariant)

Lorsqu'on crée un nouveau `Document`, on veut s'assurer qu'il soit `valide?` = tous les champs obligatoires sont présents et l'année est un nombre

4.2 Un `module DocumentBase` partagé par les exemples `Ruby`

Module `DocumentBase`

Ce module sera utilisé comme *mixin* — avec `include` — par toutes les versions de la classe `Document` que nous verrons

```ruby
module DocumentBase
  def to_s
    ed = @editeur ? ", #{@editeur}" : ''
    "\#<'#{@titre}', #{@auteurs.join(' et ')}#{ed}, #{annee}>"
  end

  def valide?
    titre && !titre.empty? &&
      annee && annee.kind_of?(Fixnum) &&
      auteurs && auteurs.kind_of?(Array) && !auteurs.empty?
  end
end
```

4.3 Diverses façons de décrire des `Documents`

Approches pour construire des descriptions de documents que nous allons examiner

Approches avec API «régulière»

- A Constructeur `new` avec arguments obligatoires seulement
- B Constructeur `new` avec arguments obligatoires/optionnels
- C Constructeur `new` avec arguments par mots-clés
- D Avec *setters*

Approches avec API «coulante»

- E API coulante avec chainage de méthodes
- F API coulante avec constructeur et bloc

Approche avec métaprogrammation

- G API coulante avec bloc et `instance_eval`

A. Constructeur `new` avec arguments obligatoires seulement

A. Constructeur `new` avec arguments obligatoires seulement : Utilisation

Exemples d'utilisation

```
Document.new( 'Domain-Specific Languages', 2011,
              ['M. Fowler', 'R. Parsons'], nil )

Document.new( 'DSLs in Action', 2011,
              ['D. Ghosh'], 'Manning' )
```

A. Constructeur `new` avec arguments obligatoires seulement : Mise en oeuvre

```
class Document
  include DocumentBase

  attr_reader :titre, :annee, :auteurs, :editeur

  def initialize( titre, annee, auteurs, editeur )
    @titre = titre
    @annee = annee
    @auteurs = auteurs
    @editeur = editeur

    DBC.assert valide?, "*** Erreur: Document mal constr
  end
end
```

A. Constructeur `new` avec arguments obligatoires seulement : Mise en oeuvre

Avantages

- Approche simple et élémentaire

Désavantages

A. Constructeur `new` avec arguments obligatoires seulement : Mise en oeuvre

Avantages

- Approche simple et élémentaire

Désavantages

- Il faut se souvenir de l'ordre des arguments ☹
- Si on a un grand nombre d'arguments optionnels, alors il faudra mettre des `nil` pour les spécifier ☹

B. Constructeur `new` avec arguments obligatoires et optionnels

B. Constructeur `new` avec arguments obligatoires et optionnels : Utilisation

Exemples d'utilisation

```
Document.new( 'Domain-Specific Languages', 2011,
              ['M. Fowler', 'R. Parsons'] )

Document.new( 'DSLs in Action', 2011,
              ['D. Ghosh'], 'Manning' )
```

B. Constructeur `new` avec arguments obligatoires et optionnels : Mise en oeuvre

```
class Document
  include DocumentBase

  attr_reader :titre, :annee, :auteurs, :editeur

  def initialize( titre, annee, auteurs, editeur = nil )
    @titre = titre
    @annee = annee
    @auteurs = auteurs
    @editeur = editeur

    DBC.assert valide?, "*** Erreur: Document mal constr
  end
end
```

Donc, plusieurs désavantages à utiliser simplement `new` pour construire des objets complexes

Avantages

- Approche simple et élémentaire

Désavantages

Donc, plusieurs désavantages à utiliser simplement `new` pour construire des objets complexes

Avantages

- Approche simple et élémentaire

Désavantages

- Il faut se souvenir de l'ordre des arguments ☹
- Si un attribut est caractérisé par une liste de valeurs, on doit utiliser explicitement un `Array` ☹

Donc, plusieurs désavantages à utiliser simplement `new` pour construire des objets complexes

Avantages

- Approche simple et élémentaire

Désavantages

- Il faut se souvenir de l'ordre des arguments ☹
- Si un attribut est caractérisé par une liste de valeurs, on doit utiliser explicitement un `Array` ☹
- Si on a plusieurs arguments optionnels indépendants, alors il faut mettre des `nil` pour les spécifier, dans le bon ordre en plus ! ☹

B. Constructeur `new` avec arguments obligatoires et optionnels : Supposons qu'on ait plusieurs options...

Par exemple, on a deux champs optionnels : `editeur` et `isbn`

Mise en oeuvre

```
class Document
  include DocumentBase
  attr_reader :titre, :annee, :auteurs, :editeur, :isbn

  def initialize( titre, annee, auteurs,
                  editeur = nil, isbn =  nil )
    @titre = titre
    @annee = annee
    @auteurs = auteurs
    @editeur = editeur
    @isbn = isbn

    DBC.assert valide?, "*** Erreur: Document mal construit"
  end
end
```

B. Constructeur `new` avec arguments obligatoires et optionnels : Supposons qu'on ait plusieurs options...

Par exemple, on a deux champs optionnels : `editeur` et `isbn`

Utilisation

```
# Document avec editeur mais sans ISBN.
Document.new( 'DSLs in Action', 2011, ['D. Ghosh'],
              'Manning' )

# Document sans editeur mais avec ISBN.
Document.new( 'DSLs in Action', 2011, ['D. Ghosh'],
              nil,
              '978-1935182450' )
```

C. Constructeur `new` avec arguments par mots-clés

C. Constructeur `new` avec arguments par mots-clés : Utilisation

Exemples d'utilisation

```
Document.new( auteurs: ['M. Fowler', 'R. Parsons'],
              titre: 'Domain-Specific Languages',
              annee: 2011 )

Document.new( annee: 2011,
              titre: 'DSLs in Action',
              auteur: 'D. Ghosh',
              editeur: 'Manning' )
```

C. Constructeur `new` avec arguments par mots-clés : Mise en oeuvre

```
class Document
  include DocumentBase

  attr_reader :titre, :annee, :auteurs, :editeur

  def initialize( titre:, annee:, auteurs: [],
                  auteur: nil, editeur: nil )
    @titre = titre
    @annee = annee
    DBC.assert !auteur || auteurs.empty?, '*** Il faut u
    @auteurs = auteurs unless auteurs.empty?
    @auteurs = [auteur] if auteur
    @editeur = editeur

    DBC.assert valide?, "*** Erreur: Document mal constr
  end
end
```

C. Constructeur `new` avec arguments par mots-clés

Avantages

- Pas besoin de se souvenir souvenir de l'ordre des arguments ☺
- Pas besoin de mettre des `nil` pour les arguments optionnels ☺
- Facile d'ajouter d'autres mots-clés pour des champs optionnels additionels ☺

Désavantages

C. Constructeur `new` avec arguments par mots-clés

Avantages

- Pas besoin de se souvenir souvenir de l'ordre des arguments ☺
- Pas besoin de mettre des `nil` pour les arguments optionnels ☺
- Facile d'ajouter d'autres mots-clés pour des champs optionnels additionels ☺

Désavantages

- Si un attribut est caractérisé par une liste de valeurs, on doit utiliser explicitement un `Array` ☹
- La méthode `initialize` peut devenir très complexe et monolithique ☹
- Pas possible dans la plupart des autres langages

D. Avec *setters*

D. Avec *setters* : Utilisation

Exemples d'utilisation

```
d1 = Document.new
d1.titre = 'Domain-Specific Languages'
d1.annee = 2011
d1.auteurs = ['M. Fowler', 'R. Parsons']

d2 = Document.new
d2.titre = 'DSLs in Action'
d2.annee = 2011
d2.auteurs = ['D. Ghosh']
d2.editeur = 'Manning'
```

D. Avec *setters* : Mise en oeuvre

```
class Document
  include DocumentBase

  attr_accessor :titre, :annee, :auteurs, :editeur

  # Par defaut, une variable d'instance est nil,
  # donc meme pas besoin de methode initialize.
end
```

D. Avec *setters* : Mise en oeuvre

Avantages

- Approche simple et élémentaire
- L'ordre des appels n'a pas d'importance
- Rien de spécial à faire pour les arguments optionnels

Désavantages

D. Avec *setters* : Mise en oeuvre

Avantages

- Approche simple et élémentaire
- L'ordre des appels n'a pas d'importance
- Rien de spécial à faire pour les arguments optionnels

Désavantages

- Comment/quand peut-on s'assurer que l'objet est construit correctement ? ☹

E. API coulante avec chainage de méthodes

E. API coulante avec chainage de méthodes : Utilisation

Exemples d'utilisation

```
Document.new
   .auteurs( 'M. Fowler', 'R. Parsons' )
   .titre( 'Domain-Specific Languages' )
   .annee( 2011 )
   .done

Document.new
   .annee( 2011 )
   .titre( 'DSLs in Action' )
   .editeur( 'Manning' )
   .auteurs( 'D. Ghosh' )
   .done
```

Le chainage de méthodes est souvent utilisé en Java

Exemple : La définition de *mocks* avec `jMock`

```
public class TimedCacheTest extends MockObjectTestCase {
    private Mock mockLoader;

    @Override
    protected void setUp() throws Exception {
        mockLoader = mock(ObjectLoader.class);
        ...
    }

    public void testLoadsObjectThatIsNotCached() {
        mockLoader.expects(once())
                  .method("load")
                  .with(eq(KEY))
                  .will(returnValue(VALUE));
        ...
    }
}
```

http://searchsoftwarequality.techtarget.com/tip/Using-JMock-in-test-driven-development

Le chainage de méthodes est souvent utilisé en Java

Exemple : Des requêtes SQL avec jOOQ

```
Author author = AUTHOR.as("author");
create
   .selectFrom(author)
   .where(exists(selectOne()
           .from(BOOK)
           .where(BOOK.STATUS.eq(BOOK_STATUS.SOLD_OUT))
           .and(BOOK.AUTHOR_ID.eq(author.ID))));
```

Source: https://en.wikipedia.org/wiki/Fluent_interface

Le chainage de méthodes est souvent utilisé en Java

Exemple : Création de documents RTF avec `jRTF`

```
rtf()
  .header(
    color( 0xff, 0, 0 ).at( 0 ),
    color( 0, 0xff, 0 ).at( 1 ),
    color( 0, 0, 0xff ).at( 2 ),
    font( "Calibri"  ).at( 0 ) )
  .section(
        p( font( 1, "Second paragraph" ) ),
        p( color( 1, "green" ) )
  )
).out( out );
```

Source: `https://dzone.com/articles/java-fluent-api-designer-crash`

E. API coulante avec chainage de méthodes : Mise en oeuvre

Retour de `self` comme résultat ⇒ permet de chainer les appels

```
class Document
  include DocumentBase

  # La methode sert a la fois de reader et de writer!
  def titre( titre = nil )
    return @titre if titre.nil?

    @titre = titre
    self
  end

  def auteurs( *auteurs )
    return @auteurs if auteurs.empty?

    @auteurs = *auteurs
    self
  end
```

E. API coulante avec chainage de méthodes : Mise en oeuvre

Retour de `self` comme résultat ⇒ permet de chainer les appels

```
class Document
  include DocumentBase

  # La methode sert a la fois de reader et de writer!
  def titre( titre = nil )
    return @titre if titre.nil?

    @titre = titre
    self
  end

  def auteurs( *auteurs )
    return @auteurs if auteurs.empty?

    @auteurs = *auteurs
    self
  end
```

E. API coulante avec chainage de méthodes : Mise en oeuvre

Retour de `self` comme résultat ⇒ permet de chainer les appels

```
class Document
  include DocumentBase

  # La methode sert a la fois de reader et de writer !
  def titre( titre = nil )
    return @titre if titre.nil?

    @titre = titre
    self
  end

  def auteurs( *auteurs )
    return @auteurs if auteurs.empty?

    @auteurs = *auteurs
    self
  end
```

E. API coulante avec chainage de méthodes : Mise en oeuvre

Retour de `self` comme résultat ⇒ permet de chainer les appels

```
class Document
  include DocumentBase

  # La methode sert a la fois de reader et de writer!
  def titre( titre = nil )
    return @titre if titre.nil?

    @titre = titre
    self
  end

  def auteurs( *auteurs )
    return @auteurs if auteurs.empty?

    @auteurs = *auteurs
    self
  end
```

E. API coulante avec chainage de méthodes : Mise en oeuvre

Retour de `self` comme résultat ⇒ permet de chainer les appels

```
class Document
  include DocumentBase

  # La methode sert a la fois de reader et de writer!
  def titre( titre = nil )
    return @titre if titre.nil?

    @titre = titre
    self
  end

  def auteurs( *auteurs )
    return @auteurs if auteurs.empty?

    @auteurs = *auteurs
    self
  end
```

E. API coulante avec chainage de méthodes : Mise en oeuvre (suite)

```ruby
def annee( annee = nil )
  return @annee if annee.nil?

  @annee = annee
  self
end

def editeur( editeur = nil )
  return @editeur if editeur.nil?

  @editeur = editeur
  self
end
```

E. API coulante avec chainage de méthodes : Mise en oeuvre (suite)

Et on définit une méthode qui effectue les validations requises

```
  def done
    DBC.assert valide?, "*** Erreur: Document mal constr

    self
  end

  alias_method :build, :done
  alias_method :__, :done
end
```

Note : La méthode `alias_method` permet d'introduire un synonyme (un *alias*) pour un nom de méthode.
Ici, les synonymes de `done` sont `build` et `__`.

E. API coulante avec chainage de méthodes : Mise en oeuvre (suite)

Et on définit une méthode qui effectue les validations requises

```
  def done
    DBC.assert valide?, "*** Erreur: Document mal constr

    self
  end

  alias_method :build, :done
  alias_method :__, :done
end
```

Note : La méthode `alias_method` permet d'introduire un synonyme (un *alias*) pour un nom de méthode.
Ici, les synonymes de `done` sont `build` et `__`.

E. API coulante avec chainage de méthodes : Utilisation (bis)

Exemples d'utilisation

```
Document.new
   .auteurs( 'M. Fowler', 'R. Parsons' )
   .titre( 'Domain-Specific Languages' )
   .annee( 2011 )
   .__

Document.new
   .annee( 2011 )
   .titre( 'DSLs in Action' )
   .editeur( 'Manning' )
   .auteurs( 'D. Ghosh' )
   .__
```

E. API coulante avec chainage de méthodes

Avantages

- L'ordre des appels n'est pas important ☺
- Facile d'ajouter de nouvelles options ☺
- Une méthode peut recevoir un nombre variable d'arguments ☺
- Mise en oeuvre simple, tant en Ruby que dans n'importe quel langage orienté objet ☺

Désavantages

E. API coulante avec chainage de méthodes

Avantages

- L'ordre des appels n'est pas important ☺
- Facile d'ajouter de nouvelles options ☺
- Une méthode peut recevoir un nombre variable d'arguments ☺
- Mise en oeuvre simple, tant en Ruby que dans n'importe quel langage orienté objet ☺

Désavantages

- On doit définir une méthode `done` ou `build` qui finalise la construction ☹
- Si on omet d'appeler cette méthode, alors la validité de l'objet ne sera pas vérifiée ☹

F. API coulante avec constructeur et bloc

F. API coulante avec constructeur et *bloc* : Utilisation

Exemples d'utilisation

```
Document.create do |d|
  d.titre = 'Domain-Specific Languages'
  d.auteurs = 'M. Fowler', 'R. Parsons'
  d.annee = 2011
end

Document.create do |d|
  d.annee = 2011
  d.titre = 'DSLs in Action'
  d.editeur = 'Manning'
  d.auteurs = 'D. Ghosh'
end
```

Est-ce que cette approche vous rappelle quelque chose ?

L'approche avec constructeur et bloc est (très !) souvent utilisée en Ruby

Spécification des commandes avec `gli`

```
command :print do |c|
  c.action do |global_options,options,args|
    motif, *fichiers = args
    fichiers << STDIN if fichiers.empty?

    fichiers.each do |fichier|
      MiniSed.traiter_fichier( fichier,
                               global_options[:in_place]
      do |flux|
        MiniSed.print( motif, flux.readlines )
      end
    end
  end
end
```

L'approche avec constructeur et bloc est (très !) souvent utilisée en Ruby

Spécification d'un `gemspec`

```
spec = Gem::Specification.new do |s|
  s.name = 'mini-sed'
  s.version = MiniSed::VERSION
  ...
  s.summary = 'Une version simplifiee de sed'
  ...
  s.add_development_dependency('minitest', '5.4.3')
  s.add_runtime_dependency('gli','2.12.0')
end
```

L'approche avec constructeur et bloc est (très !) souvent utilisée en Ruby

Spécification de certaines tâches dans un `Rakefile`

```
desc 'Run features tagged as work-in-progress (@wip)'
Cucumber::Rake::Task.new('features:wip') do |t|
  ...
  t.cucumber_opts = "features
                     --format html -o #{CUKE_RESULTS}
                     --format progress -x --no-color
                     -s#{tag_opts}"
  t.fork = false
end
```

L'approche avec constructeur et bloc est (très !) souvent utilisée en Ruby

Spécification de tables relationnelles en `Rails`

```
ActiveRecord::Schema.define(version: 20151109232424) do
  create_table "orders", force: true do |t|
    t.integer  "product_id"
    t.integer  "quantity"
    t.datetime "created_at"
    t.datetime "updated_at"
  end

  add_index "orders", ["product_id"],
            name: "index_orders_on_product_id"

  ...
end
```

F. API coulante avec constructeur et *bloc* : Mise en oeuvre

```
class Document
  include DocumentBase
  attr_reader :titre, :annee, :auteurs, :editeur
  attr_writer :titre, :annee, :editeur

  def self.create
    nouveau_doc = new
    yield nouveau_doc
    DBC.assert nouveau_doc.valide?, "*** Erreur: Documen
    nouveau_doc
  end

  def auteurs=( auteurs )
    @auteurs =
      auteurs.class == Array ? auteurs : [auteurs]
  end

  private_class_method :new
end
```

F. API coulante avec constructeur et *bloc* : Mise en oeuvre

```
class Document
  include DocumentBase
  attr_reader :titre, :annee, :auteurs, :editeur
  attr_writer :titre, :annee, :editeur

  def self.create
    nouveau_doc = new
    yield nouveau_doc
    DBC.assert nouveau_doc.valide?, "*** Erreur: Documen
    nouveau_doc
  end

  def auteurs=( auteurs )
    @auteurs =
      auteurs.class == Array ? auteurs : [auteurs]
  end

  private_class_method :new
end
```

F. API coulante avec constructeur et *bloc* : Mise en oeuvre

```
class Document
  include DocumentBase
  attr_reader :titre, :annee, :auteurs, :editeur
  attr_writer :titre, :annee, :editeur

  def self.create
    nouveau_doc = new
    yield nouveau_doc
    DBC.assert nouveau_doc.valide?, "*** Erreur: Documen
    nouveau_doc
  end

  def auteurs=( auteurs )
    @auteurs =
      auteurs.class == Array ? auteurs : [auteurs]
  end

  private_class_method :new
end
```

F. API coulante avec constructeur et *bloc* : Mise en oeuvre (suite)

Appel du *builder*

```
Document.create do |d|
  d.titre = 'Domain-Specific Languages'
  d.auteurs = 'M. Fowler', 'R. Parsons'
  d.annee = 2011
end
```

Définition du *builder* : On évalue le bloc passé à la méthode `create`, en donnant en argument le nouvel objet créé par `new`

```
def self.create
  nouveau_doc = new

  yield nouveau_doc

  DBC.assert nouveau_doc.valide?, "*** Erreur: Documen

  nouveau_doc
end
```

G. API coulante avec bloc et `instance_eval`

G. API coulante avec bloc et `instance_eval` : Utilisation

Exemples d'utilisation

```
Document.create do
  annee 2011
  titre 'Domain-Specific Languages'
  auteurs 'M. Fowler', 'R. Parsons'
end

Document.create do
  titre 'DSLs in Action'
  auteurs 'D. Ghosh'
  annee 2011
  editeur 'Manning'
end
```

L'approche avec bloc et `instance_eval` est aussi utilisée fréquemment en Ruby

Spécification de tests unitaire pour des applications Web avec `Capybara`

```
it "permet a un enseignant de se logger, de creer une bo
  visit "/connexion"
  fill_in "Identifiant", :with => ENSEIGNANT
  fill_in "Mot de passe", :with => MP_ENSEIGNANT
  choose "groupeEnseignant"
  click_button "Connexion"
  expect(page).to have_content "Bienvenue sur l'applicat
  expect(page).to have_content "enseignant"

  ...

  click_link "Se deconnecter"
  expect(page).to have_content "Pour utiliser Oto, vous
end
```

Source: `http://www.rubydoc.info/gems/factory_girl/file/GETTING_STARTED.md`

L'approche avec bloc et `instance_eval` est aussi utilisée fréquemment en Ruby

Spécification d'applications Web avec `Sinatra`

```
class DocumentedApp < Sinatra::Base
  register Sinatra::DocDsl

  page do
    title "DocDSL demo"
    header "Displayed before the title."
    introduction "A short introduction to your API."
    url_prefix "/my/application/path"

    footer " ... "
    ...
    configure_renderer do
      self.render_md  # Mark down
    end
  end
end
```

Source: `https://github.com/jillesvangurp/sinatra-docdsl`

L'approche avec bloc et `instance_eval` est aussi utilisée fréquemment en Ruby

Spécification d'objets pour des tests d'applications Rails avec `factory_girl`

```
FactoryGirl.define do
  factory :user do
    first_name "John"
    last_name  "Doe"
    admin false
  end

  # This will use the User class (Admin would have been
  factory :admin, class: User do
    first_name "Admin"
    last_name  "User"
    admin      true
  end
end
```

Source: `http://www.rubydoc.info/gems/factory_girl/file/GETTING_STARTED.md`

L'approche avec bloc et `instance_eval` est aussi utilisée fréquemment en Ruby

Spécification d'objets pour des tests avec `machinist` en Rails

```
Post.blueprint do
  author
  title  { "Post #{sn}" }
  body   { "Lorem ipsum..." }
end

Comment.blueprint do
  post
  email { "commenter#{sn}@example.com" }
  body  { "Lorem ipsum..." }
end
```

Source: `http://www.rubydoc.info/gems/factory_girl/file/GETTING_STARTED.md`

G. API coulante avec bloc et `instance_eval` : Mise en oeuvre

```
class Document
  include DocumentBase

  def titre( titre = nil )
    return @titre if titre.nil?

    @titre = titre
  end

  def auteurs( *auteurs )
    return @auteurs if auteurs.empty?
    @auteurs = *auteurs
  end

  def annee( annee = nil )
    return @annee if annee.nil?
    @annee = annee
  end
```

G. API coulante avec bloc et `instance_eval` : Mise en oeuvre

```ruby
class Document
  include DocumentBase

  def titre( titre = nil ) # reader
    return @titre if titre.nil?

    @titre = titre
  end

  def auteurs( *auteurs )
    return @auteurs if auteurs.empty?
    @auteurs = *auteurs
  end

  def annee( annee = nil )
    return @annee if annee.nil?
    @annee = annee
  end
```

G. API coulante avec bloc et `instance_eval` : Mise en oeuvre

```
class Document
  include DocumentBase

  def titre( titre = nil ) # writer
    return @titre if titre.nil?

    @titre = titre
  end

  def auteurs( *auteurs )
    return @auteurs if auteurs.empty?
    @auteurs = *auteurs
  end

  def annee( annee = nil )
    return @annee if annee.nil?
    @annee = annee
  end
```

G. API coulante avec bloc et `instance_eval` : Mise en oeuvre (suite)

```
  def editeur( editeur = nil )
    return @editeur if editeur.nil?
    @editeur = editeur
  end

  def self.create( &bloc )
    nouveau_doc = new
    nouveau_doc.instance_eval &bloc
    DBC.assert nouveau_doc.valide?, "*** Erreur: Documen

    nouveau_doc
  end

  private_class_method :new
end
```

Ce que fait `instance_eval` : Évalue un segment de code dans le contexte du récepteur du message

Description de `instance_eval`

```
instance_eval { |obj| block } -> obj
```

Evaluates [. . .] the given `block` *within the context of the receiver* `(obj)`*. In order to set the context, the variable* `self` *is set to* `obj` *while the code is executing, giving the code access to* `obj`*'s instance variables and private methods.*

Ce que fait `instance_eval` : Évalue un segment de code dans le contexte du récepteur du message

Un petit exemple

Définition d'une classe `A`

```
class A
  def initialize( x )
    @x = x
  end

  def val
    @x
  end
  private :val

  def plus( y )
    @x + y
  end
end
```

Quelques appels

```
a = A.new( 10 )

puts a.instance_eval { @x }
=> 10

puts a.instance_eval { plus 2 }
=> 12

puts a.val
=> NoMethodError: private method
   'val' called for
   #<A:0x13969fbe @x=10>

puts a.instance_eval { val }
=> 10
```

G. API coulante avec bloc et `instance_eval` : Mise en oeuvre (suite)

Appel du *builder*

```
Document.create do
  annee 2011
  titre 'Domain-Specific Languages'
  auteurs 'M. Fowler', 'R. Parsons'
end
```

Définition du *builder* ⇒ Évaluation de code

```
def self.create( &bloc )
  nouveau_doc = new

  nouveau_doc.instance_eval &bloc

  DBC.assert nouveau_doc.valide?, "*** Erreur: Document mal

  nouveau_doc
end
```

4.4 Deux API coulantes en Java pour décrire des `Document`s

Certaines constructions `Java` permettent de définir des API coulantes semblables à celles en Ruby

Les lambda-expressions

Construction relativement nouvelle : Présente depuis Java 8.0

Les classes internes anonymes et les *initializers*

Constructions pas nouvelles. . . mais peu connues

A. API coulante avec lambda-expression

A. API coulante avec lambda-expression : Utilisation

```
Document d = new Document( doc -> {
    doc.title( "Domain-Specific Languages" );
    doc.authors( "M. Fowler", "R. Parsons" );
    doc.publisher( "Pearsons Education" );
    doc.year( 2011 );
  } );
```

A. API coulante avec lambda-expression : Mise en oeuvre

```
interface DocumentBuilder { void build( Document d ); }

class Document {
  private String title, publisher;
  private String[] authors;
  private int year;

  Document( DocumentBuilder b ) {
    b.build( this );
  }

  protected void title( String t ) { title = t; }

  protected void authors( String... auths ) {
    authors = auths.clone();
  }
  ...
```

A. API coulante avec lambda-expression : Mise en oeuvre (suite)

```
...
protected void publisher( String p ) {
  publisher = p;
}

protected void year( int y ) {
  year = y;
}


...
public String title() {
  return title;
}

public String[] authors() {
  return authors;
}
...
```

A. API coulante avec lambda-expression

Utilisation

```
Document d = new Document( doc -> {
    doc.title( "Domain-Specific Languages" );
    doc.authors( "M. Fowler", "R. Parsons" );
    doc.publisher( "Pearsons Education" );
    doc.year( 2011 );
  } );
```

Mise en oeuvre du constructeur `Document`

```
Document( DocumentBuilder b ) {
  b.build( this );
}
```

Donc : le constructeur `Document` appelle la lambda-expression avec l'objet nouvellement alloué comme argument.

B. API coulante avec classe interne anonyme et *initializer*

B. API coulante avec classe interne anonyme et *initializer* : Utilisation

Utilisation

```
Document d = new Document() {{
      title( "Domain-Specific Languages" );
      authors( "M. Fowler", "R. Parsons" );
      publisher( "Pearsons Education" );
      year( 2011 );
    }};
```

B. API coulante avec classe interne anonyme et *initializer* : Mise en oeuvre

```
class Document {
  private String title, publisher;
  private String[] authors;
  private int year;

  Document() {}

  protected void title( String t ) { title = t; }

  protected void authors( String... auths ) {
    authors = auths.clone();
  }
  ...
```

B. API coulante avec classe interne anonyme et *initializer*

Utilisation

```
Document d = new Document() {{
      title( "Domain-Specific Languages" );
      authors( "M. Fowler", "R. Parsons" );
      publisher( "Pearsons Education" );
      year( 2011 );
    }};
```

Mise en oeuvre du constructeur `Document`

```
Document() {}
```

B. API coulante avec classe interne anonyme et *initializer*

Utilisation

```
Document d = new Document() {{
      title( "Domain-Specific Languages" );
      authors( "M. Fowler", "R. Parsons" );
      publisher( "Pearsons Education" );
      year( 2011 );
    }};
```

- Premier niveau de bloc : Classe interne anonyme
- Deuxième niveau de bloc : *instance initializer*

Une classe interne anonyme associée à l'objet peut redéfinir des méthodes de l'objet

```
class Foo {
  Foo() {}

  protected int val;
  public int val() { return val; }
  public void setVal( int v ) { val = v; }
}
---
Foo c = new Foo() {
     public void setVal( int v ) { x = 10 * v + 1; }
};

c.setVal( 12 );
System.out.println( c.val() ); // =>   ?
```

Une classe interne anonyme associée à l'objet peut redéfinir des méthodes de l'objet

```
class Foo {
  Foo() {}

  protected int val;
  public int val() { return val; }
  public void setVal( int v ) { val = v; }
}
---
Foo c = new Foo() {
     public void setVal( int v ) { x = 10 * v + 1; }
};

c.setVal( 12 );
System.out.println( c.val() ); // => 121
```

L'initialisation d'un nouvel objet peut se faire dans un bloc d'initialisation d'instance

```
class Foo {
  Foo() {}

  protected int val;
  public int val() { return val; }
  public void setVal( int v ) { val = v; }

  { val = 99; }
}
---
Foo c = new Foo();

System.out.println( c.val() ); // => ?
```

L'initialisation d'un nouvel objet peut se faire dans un bloc d'initialisation d'instance

```
class Foo {
  Foo() {}

  protected int val;
  public int val() { return val; }
  public void setVal( int v ) { val = v; }

  { val = 99; }  // Initialisation d'instance.
}
---
Foo c = new Foo();

System.out.println( c.val() ); // => 99
```

Remarque : On peut aussi faire d'autres choses «bizarres» en Java — c'est juste plus compliqué ☹

Une classe `Foo`

```
class Foo {
  private int val;

  Foo( int val ) {
    this.val = val;
  }

  private void inc() {
    val += 1;
  }

  int val() {
    return val;
  }
}
```

Remarque : On peut aussi faire d'autres choses «bizarres» en Java — c'est juste plus compliqué ☹

Qu'imprime ce programme ?

```
class CallFoo {
  public static void main( String args[] ) {
    Foo f = new Foo(99);
    f.inc();
  }
}
```

Remarque : On peut aussi faire d'autres choses «bizarres» en Java — c'est juste plus compliqué ☹

Qu'imprime ce programme ?

```
class CallFoo {
  public static void main( String args[] ) {
    Foo f = new Foo(99);
    f.inc();
  }
}
```

Erreur de compilation ☹

```
$ javac CallFoo.java
CallFoo.java:4: error: inc() has private access in Foo
    f.inc();
     ^
1 error
```

Remarque : On peut aussi faire d'autres choses «bizarres» en Java — c'est juste plus compliqué ☹

Qu'imprime ce programme (bis) ?

```
class CallFoo
  public static void main( String args[] ) {
    Foo f = new Foo(99);
    try {  java.lang.reflect.Method m =
                 Foo.class.getDeclaredMethod("inc");
           m.setAccessible(true);
           m.invoke(f);
    } catch( Throwable t ) {}
    System.out.println( "val = " + f.val() );
  }
}
```

Remarque : On peut aussi faire d'autres choses «bizarres» en Java — c'est juste plus compliqué ☹

Qu'imprime ce programme (bis) ?

```
class CallFoo
  public static void main( String args[] ) {
    Foo f = new Foo(99);
    try {  java.lang.reflect.Method m =
                Foo.class.getDeclaredMethod("inc");
           m.setAccessible(true);
           m.invoke(f);
    } catch( Throwable t ) {}
    System.out.println( "val = " + f.val() );
  }
}
```

Il appelle la méthode privée `inc()`, donc...

```
val = 100
```

5. Conclusion

Ruby est un langage puissant et expressif

Ruby is “A dynamic, open source programming language with a focus on simplicity and productivity.”

Source: `https://www.ruby-lang.org/en/`

Ruby facilite la définition de DSLs internes expressifs

- Syntaxe flexible
- Argument par mots-clés
- Blocs et fermetures
- Métaprogrammation dynamique

Mais on peut définir des DSLs internes dans n'importe quel langage… même Java

Constructeurs et chainage de méthodes

Lambda-expressions

Annotations

Les DSLs peuvent simplifier l'expression de solutions à de nombreux problèmes

La seule limite...

Les DSLs peuvent simplifier l'expression de solutions à de nombreux problèmes

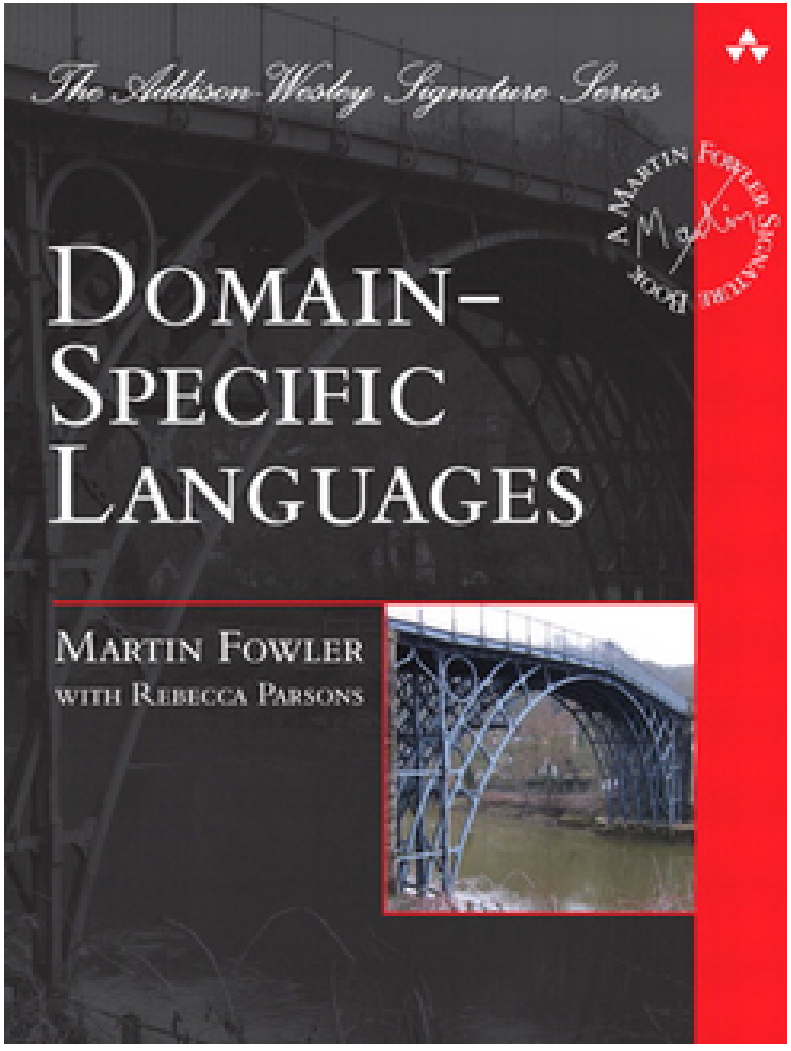
La seule limite pour le Projet #3

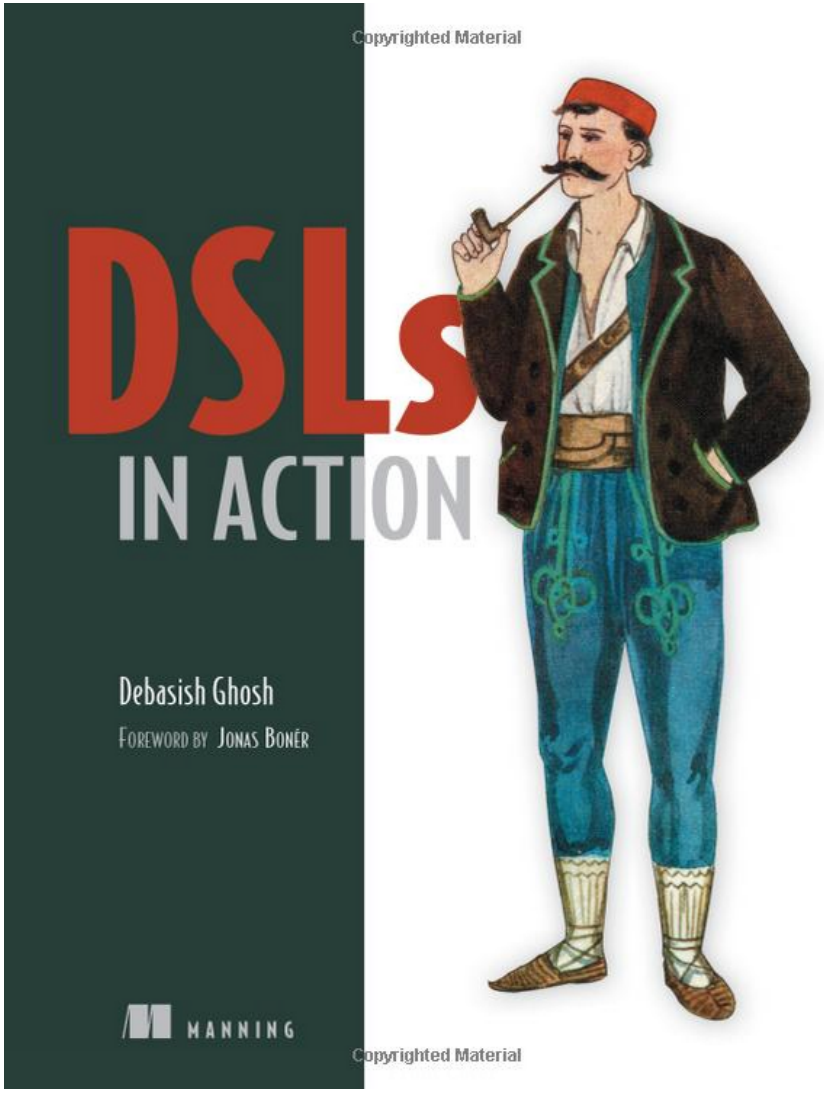
Les DSLs peuvent simplifier l'expression de solutions à de nombreux problèmes

La seule limite pour le Projet #3

Votre imagination !

Pour des informations additionnelles sur les DSLs





Pour des informations additionnelles sur les DSLs

M. Fowler.

A pedagogical framework for domain-specific languages.

IEEE Software, 26(4) :13–14, July 2009.

M. Mernik, J. Heering, and A.M. Sloane.

When and how to develop domain-specific languages.

ACM Comput. Surv., 37(4) :316–344, 2005.

M. Voelter.

DSL Engineering : Designing, Implementing and Using Domain-Specific Languages.

Createspace, 2013.

`dslbook.org`.