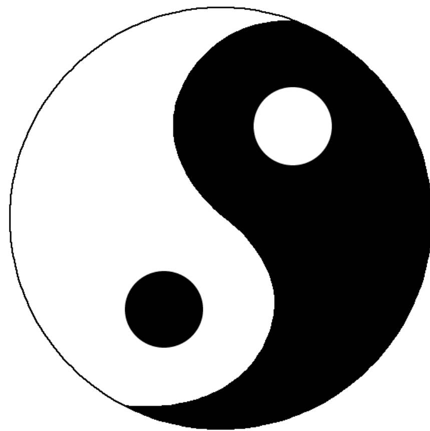
Domain-Specific Language Definition Through Reflective Extensible Language Kernels

Sebastián González and Wolfgang De Meuter

September 18, 2003

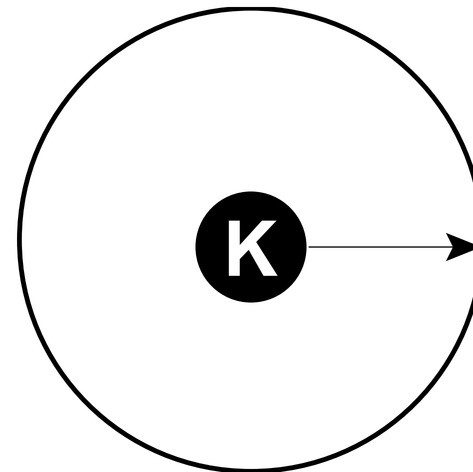
The problem

A frequent dilemma in programming language design is the choice between a language with a rich set of notations and features, and a small, simple core language.



The solution

We address the dilemma by proposing an extensible language kernel. The kernel's syntax & semantics can be grown as needed by means of reflection.



Configurable grammar

<expression> ||= <comparison>

<comparison> ::= <addition> (<comparison_op> <addition>) ∗

<comparison_op> ||= ⊥

<addition> ::= <product> (<addition_op> <product>) ∗

<addition_op> ||= ⊥

<product> ::= <power> (<product_op> <power>) ∗

<product_op> ||= ⊥

<power> ::= <operand> (<power_op> <operand>) ∗

<power_op> ||= ⊥

<operand> ||= ≺string≻ | ≺number≻ | ... | <table> | <application>

<application> ::= ≺reference≻ " @ " <operand>

<table> ::= " [" (<expression> " , ") ∗ <expression> | ≺success≻ "] "

Extending the grammar

Initially, only pure functional @-notation is allowed:

```
binary_sub@[ binary_sum@[ 1, 2 ], 3 ]
```

If the grammar is extended:

<addition_op> ||= "+" | "-"

<addition> ::= <product> (<addition_op> <product>) *

Then expressions like the following become valid:

```
1+2-3
```

An example: introducing *ex-nihilo* object creation

```
john: {-
  getname():: "John Coltraine";
  getage():: "40" -}
```

```
install_syntax(operand_parser, "ex-nihilo object block",
  void, var_mixfix_operator("{-", ";", "-}"))

ex_nihilo_creation@'objdef:- {
  object: dictionary(parent());
  eval(unquote(objdef), object);
  object }

assign_semantics("ex-nihilo object block",
  ex_nihilo_creation)
```