



Extensibility

Didier Verna

DSL Overview

Taxonomy

Extensibility

Example

Internal

External

Conclusion

Discussion

DSLs from the Perspective of Extensible Languages

Didier Verna

didier@lrde.epita.fr
<http://www.lrde.epita.fr/~didier>

ACCU 2012 – Thursday, April 26th



Introduction

The Challenges in DSL design and implementation

Extensibility

Didier Verna

DSL Overview

Taxonomy

Extensibility

Example

Internal

External

Conclusion

Discussion

- Orthogonal expertise
 - ▶ Application domain
 - ▶ Language design and implementation
- DSLs vs GPLs
 - ▶ $\neq$ syntax
 - ▶ $\neq$ semantics
- ▶ DSLs and GPLs need to be completely different.

Really ?



Table of contents

Extensibility

Didier Verna

DSL Overview

Taxonomy

Extensibility

Example

Internal

External

Conclusion

Discussion

- 1 DSL Overview
 - Taxonomy of DSLs
 - Extensibility at a glance

- 2 A concrete example
 - Building the DSL
 - Externalizing it

- 3 Conclusion
 - Discussion



Taxonomy of DSLs

[Fowler, 2005, Tratt, 2008]

Extensibility

Didier Verna

DSL Overview

Taxonomy

Extensibility

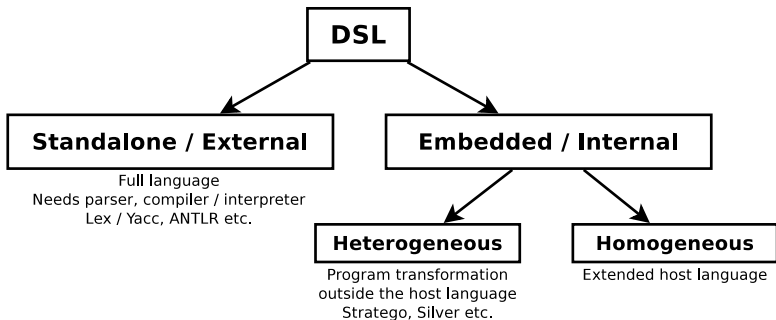
Example

Internal

External

Conclusion

Discussion





Extensibility at a glance I

[van Deursen et al., 2000, Vasudevan and Tratt, 2011]

Extensibility

Didier Verna

DSL Overview

Taxonomy

Extensibility

Example

Internal

External

Conclusion

Discussion

- User-defined data types and (infix) operators
Algol 68 [Denert et al., 1975, Pagan, 1979]
- Operator overloading
C++ [McNamara and Smaragdakis, 2000]
- Compile-Time Meta-Programming
C++ templates [Prud'homme, 2006]
Template Haskell, Meta OCaml [Czarnecki et al., 2004]
Meta Lua, Converge, Nermerle [Tratt, 2005,
Fleutot and Tratt, 2007, Skalski et al., 2004]. . .
- Functional languages (Haskell, ML)
[Kamin, 1998, Elliott, 1999, Hudak, 1998]



Extensibility at a glance II

[van Deursen et al., 2000, Vasudevan and Tratt, 2011]

Extensibility

Didier Verna

DSL Overview

Taxonomy

Extensibility

Example

Internal

External

Conclusion

Discussion

- Forth: Operator extensibility + CTMP
[Ahson and Lamba, 1985]
- Scala [Rompf et al., 2011]
- Marginally: $\text{T}_{\text{E}}\text{X}$, m4



Example

Command-line options highlighting

Extensibility

Didier Verna

DSL Overview

Taxonomy

Extensibility

Example

Internal

External

Conclusion

Discussion

Usage: advanced **`[-hd] [+d]`** [OPTIONS] cmd [OPTIONS]

Available commands: push pull.

Use 'cmd --help' to get command-specific help.

`-h, --help` Print this help and exit.

`-(+)d, --debug[=on/off]` Turn debugging on or off.

Fallback: on

Environment: DEBUG

- Properties (bold, underline, foreground color...)
- Faces (localized property set)
- Themes (face trees)



Step 1: the basic thing

Extensibility

Didier Verna

DSL Overview

Taxonomy

Extensibility

Example

Internal

External

Conclusion

Discussion

```
(setq default-theme
  (make-instance 'face :name 'toplevel
    :background 'black
    :subfaces (list (make-instance 'face :name 'option
      :foreground 'white
      :subfaces (list (make-instance 'face :name 'syntax
        :bold t
        :foreground 'cyan)
        (make-instance 'face :name 'usage
          :foreground 'yellow)))))))
```

Problems with `make-instance`:

- 1 Class name exposed
- 2 Name argument optional



Step 2: an instantiation wrapper

Extensibility

Didier Verna

DSL Overview

Taxonomy

Extensibility

Example

Internal

External

Conclusion

Discussion

```
(setq default-theme
  (make-face 'toplevel
    :background 'black
    :subfaces (list (make-face 'option
      :foreground 'white
      :subfaces (list (make-face 'syntax
        :bold t
        :foreground 'cyan)
        (make-face 'usage
          :foreground 'yellow))))))
```

Problems:

- 1 Explicit `toplevel` face name
- 2 Explicit creation of sub-faces *list*
- 3 Lots of calls to `make-face`
- 4 Lots of quoting



Step 3: a theme creation wrapper

Solution to problem #1

Extensibility

Didier Verna

DSL Overview

Taxonomy

Extensibility

Example

Internal

External

Conclusion

Discussion

```
(setq default-theme
  (make-theme
    :background 'black
    :subfaces (list (make-face 'option
                          :foreground 'white
                          :subfaces (list (make-face 'syntax
                                                    :bold t
                                                    :foreground 'cyan)
                                              (make-face 'usage
                                                    :foreground 'yellow)))))
```

Problems:

- 1 ~~Explicit toplevel face name~~
- 2 Explicit creation of sub-faces *list*
- 3 Lots of calls to `make-face`
- 4 Lots of quoting



Step 4: CLOS and the MOP

Solution to problem #2

Extensibility

Didier Verna

DSL Overview

Taxonomy

Extensibility

Example

Internal

External

Conclusion

Discussion

Generic functions, methods

```
(defmethod func ((arg1 class1) arg2 ...)  
  body)
```

- ▶ **Methods are *outside* the classes** (ordinary function calls)
- ▶ **Multiple dispatch** (multi-methods)

The CLOS Meta-Object Protocol (MOP)

■ **CLOS *itself* is object-oriented**

- ▶ The CLOS MOP: a *de facto* implementation standard
 - ▶ The CLOS components (classes *etc.*) are (meta-)objects of some (meta-)classes
- ▶ **`initialize-instance` is a generic function**



Step 4: CLOS and the MOP

Solution to problem #2

Extensibility

Didier Verna

DSL Overview

Taxonomy

Extensibility

Example

Internal

External

Conclusion

Discussion

```
(setq default-theme
  (make-theme
    :background 'black
    :face (make-face 'option
      :foreground 'white
      :face (make-face 'syntax :bold t :foreground 'cyan)
      :face (make-face 'usage :foreground 'yellow))))
```

Problems:

- 1 ~~Explicit toplevel face name~~
- 2 ~~Explicit creation of sub-faces *list*~~
- 3 Lots of calls to `make-face`
- 4 Lots of quoting



Step 5: syntax extension

Solution to problem #3

Extensibility

Didier Verna

DSL Overview

Taxonomy

Extensibility

Example

Internal

External

Conclusion

Discussion

- *readtable*: currently active syntax extensions table
- *macro character*: special syntactic meaning
- *reader macro*: implements macro character behavior



Step 5: syntax extension

Solution to problem #3

Extensibility

Didier Verna

DSL Overview

Taxonomy

Extensibility

Example

Internal

External

Conclusion

Discussion

```
(setq default-theme
  (make-theme
    :background 'black
    :face { 'option :foreground 'white
            :face { 'syntax :bold t :foreground 'cyan }
            :face { 'usage :foreground 'yellow }
          }
  ))
```

Problems:

- 1 ~~Explicit toplevel face name~~
- 2 ~~Explicit creation of sub-faces *list*~~
- 3 ~~Lots of calls to `make-face`~~
- 4 ~~Lots of quoting~~



Step 6: macros

Solution to problem #4

Extensibility

Didier Verna

DSL Overview

Taxonomy

Extensibility

Example

Internal

External

Conclusion

Discussion

- Ordinary Lisp functions
- Work on chunks of code (as data)
- Transform expressions into a new expression
- Compile-time effect
- Control over evaluation



Step 6: macros

Solution to problem #4

Extensibility

Didier Verna

DSL Overview

Taxonomy

Extensibility

Example

Internal

External

Conclusion

Discussion

```
(setq default-theme
  (define-theme
    :background black
    :face { option :foreground white
              :face { syntax :bold t :foreground cyan }
              :face { usage :foreground yellow }
            }
  ))
```

Problems:

- 1 ~~Explicit toplevel face name~~
- 2 ~~Explicit creation of sub-faces *list*~~
- 3 ~~Lots of calls to `make-face`~~
- 4 ~~Lots of quoting~~



Final result

Extensibility

Didier Verna

DSL Overview

Taxonomy

Extensibility

Example

Internal

External

Conclusion

Discussion

Looks pretty much like a DSL to me...

```
;; My personal theme with so cool colors
```

```
:background black
:face {option :foreground white
      :face {syntax :bold t :foreground cyan}
      :face {usage :foreground yellow}}
```

■ read, eval and possibly compile



Conclusion

Extensibility

Didier Verna

DSL Overview

Taxonomy

Extensibility

Example

Internal

External

Conclusion

Discussion

- Impact of GPL on DSL design and implementation
- Key GPL aspect: extensibility
- Embedded homogeneous approach
 - ▶ A single language
 - ▶ DSL infrastructure smaller
 - ▶ DSL both internal and external
- Common Lisp
 - ▶ Functional, Imperative, Object-Oriented
 - ▶ MOP
 - ▶ CTMP
 - ▶ Syntax extension
 - ▶ `read`, `eval`, `compile`



Internal vs External DSLs

[Kamin, 1998, Czarnecki et al., 2004]

Extensibility

Didier Verna

DSL Overview

Taxonomy

Extensibility

Example

Internal

External

Conclusion

Discussion

■ Suboptimal syntax **ok but...**

Not ok:

- ▶ [Fowler, 2010]: “external DSLs have their own custom syntax and you write a full parser to process them”
- ▶ [Kamin, 1998, Czarnecki et al., 2004]: “a prerequisite for embedding is that the syntax for the new language be a subset of the syntax for the host language”
- ▶ BTW, same disagreement at the semantic level (MOP)

■ Poor error reporting

- ▶ Research: [Tratt, 2008]
- ▶ Lisp: ? (but Cf. condition system & restarts)



Controversial aspects of extensibility

Extensibility

Didier Verna

DSL Overview

Taxonomy

Extensibility

Example

Internal

External

Conclusion

Discussion

■ Dynamic typing

- ▶ pros: end-user friendly
- ▶ cons: run-time type errors / checking
- ▶ Research: [Taha and Sheard, 1997]
- ▶ Hybrid languages (Cf. Racket)

■ Lazy Evaluation

- ▶ pros: infinite data structures, new control primitives *etc.*
- ▶ cons: pure functional languages only
- ▶ Lisp: laziness through macros (not as straightforward), but side-effects for free, and still functional.



The root of (Lisp) extensibility

Extensibility

Didier Verna

DSL Overview

Taxonomy

Extensibility

Example

Internal

External

Conclusion

Discussion

■ Reflexion

- ▶ Introspection
- ▶ Intercession

■ Implementation

- ▶ By API
- ▶ Inherent: “homoiconicity” [McIlroy, 1960, Kay, 1969]

■ Further distinction [Maes, 1987, Smith, 1984]

- ▶ Structural reflexion (program)
- ▶ Behavioral reflexion (language)



Bibliography I

Extensibility

Didier Verna



Ahson, S. I. and Lamba, S. S. (1985).

The use of Forth language in process control.

Computer Languages, 10:179–187.



Czarnecki, K., O'Donnell, J., Striegnitz, J., and Taha, W. (2004).

DSL implementation in MetaOCaml, Template Haskell, and C++.

In Lengauer, C., Batory, D., Consel, C., and Odersky, M., editors, *Domain-Specific Program Generation*, volume 3016 of *Lecture Notes in Computer Science*, pages 51–72. Springer Berlin / Heidelberg.



Bibliography II

Extensibility

Didier Verna



Denert, E., Ernst, G., and Wetzel, H. (1975).
Graphex68 graphical language features in algol 68.
Computers & Graphics, 1(2-3):195–202.



van Deursen, A., Klint, P., and Visser, J. (2000).
Domain-specific languages: an annotated bibliography.
SIGPLAN Notices, 35:26–36.



Elliott, C. (1999).
An embedded modeling language approach to
interactive 3D and multimedia animation.
IEEE Transactions on Software Engineering,
25:291–308.



Bibliography III

Extensibility

Didier Verna



Fleutot, F. and Tratt, L. (2007).

Contrasting compile-time meta-programming in Metalua and Converge.

In Workshop on Dynamic Languages and Applications.



Fowler, M. (2005).

Language workbenches: The killer-app for domain specific languages?



Fowler, M. (2010).

Domain Specific Languages.
Addison Wesley.



Bibliography IV

Extensibility

Didier Verna



Hudak, P. (1998).

Modular domain specific languages and tools.

In Proceedings of the 5th International Conference on Software Reuse, ICSR'98, pages 134–142, Washington, DC, USA. IEEE Computer Society.



Kamin, S. N. (1998).

Research on domain-specific embedded languages and program generators.

In Electronic Notes in Theoretical Computer Science, volume 14. Elsevier.



Kay, A. C. (1969).

The Reactive Engine.

PhD thesis, University of Hamburg.



Bibliography V

Extensibility

Didier Verna



Maes, P. (1987).

Concepts and experiments in computational reflection.
In OOPSLA. ACM.



McIlroy, M. D. (1960).

Macro instruction extensions of compiler languages.
Commun. ACM, 3:214–220.



McNamara, B. and Smaragdakis, Y. (2000).

Functional programming in C++.
SIGPLAN Not., 35:118–129.



Pagan, F. (1979).

ALGOL 68 as a metalanguage for denotational semantics.
The Computer Journal, 22(1):63–66.



Bibliography VI

Extensibility

Didier Verna



Prud'homme, C. (2006).

A domain specific embedded language in c++ for automatic differentiation, projection, integration and variational formulations.

Journal of Scientific Programming, 14:81–110.



Rompf, T., Sujeeth, A. K., Lee, H., Brown, K. J., Chafi, H., Odersky, M., and Olukotun, K. (2011).

Building-blocks for performance oriented DSLs.

In DSL'11: IFIP Working Conference on Domain-Specific Languages.



Skalski, K., Moskal, M., and Olszta, P. (2004).

Meta-programming in Nemerle.

Technical report.



Bibliography VII

Extensibility

Didier Verna



Smith, B. C. (1984).

Reflection and semantics in Lisp.

In *Symposium on Principles of Programming Languages*, pages 23–35. ACM.



Taha, W. and Sheard, T. (1997).

Multi-stage programming with explicit annotations.

In *Proceedings of the 1997 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*, PEPM '97, pages 203–217, New York, NY, USA. ACM.



Bibliography VIII

Extensibility

Didier Verna



Tratt, L. (2008).

Domain specific language implementation via compile-time meta-programming.

ACM Transactions on Programming Languages and Systems, 30:31:1–31:40.



Tratt, L. (2005).

Compile-time meta-programming in a dynamically typed OO language.



Vasudevan, N. and Tratt, L. (2011).

Comparative study of DSL tools.

Electronic Notes in Theoretical Computer Science, 264:103–121.