Building Domain Specific Languages with the Metacasanova meta-compiler

Francesco Di Giacomo

Università Ca' Foscari di Venezia - PhD in Computer Science

- They allow to express the solution of a problem in a more natural way.
- They provide constructs that are domain-specific not provided by GPL's.
- They allow to develop complete application programs for a specific domain more quickly.

Introduction

Importance of domain specific languages

- They allow to express the solution of a problem in a more natural way.
- They provide constructs that are domain-specific not provided by GPL's.
- They allow to develop complete application programs for a specific domain more quickly.

CONSEQUENCE: It is desirable to deploy a DSL's when the scope of the application is very specific.

Introduction

Some DSL's examples

DSL	**Application**
Lex and Yacc	program lexing and parsing
PERL	text/file manipulation/scripting
VHDL	hardware description
TeX, LaTeX, troff	document layout
HTML, SGML	document markup
SQL, LDL, QUEL	databases
pic, postscript	2D graphics
Open GL	high-level 3D graphics
Tcl, Tk	GUI scripting
Mathematica, Maple	symbolic computation
AutoLisp/AutoCAD	computer aided design
Csh	OS scripting (Unix)
IDL	component technology (COM/CORBA)
Emacs Lisp	text editing
Prolog	logic
Visual Basic	scripting and more
Excel Macro Language	spreadsheets and many things never intended

PROBLEM: Creating DSL's requires to implement a compiler/interpreter for the language.

- Compilers are complex
- Several modules: parser, type checker, code generator/code interpreter
- Require a lot of development time.
- Not flexible: adding features to the language compiled by hard-coded compilers takes a considerable effort.

Towards meta-compilers

Implementing compilers is repetitive

The implementation of the compiler is a repetitive process:

- The parser can be created with parser generators (e.g. Yacc)
- The type system must be implemented in the host language.
- The operational semantics must be reflected in the generated code (code generations).

Towards meta-compilers

Type system and semantics

How they are formalized:

- Expressed in a form that mimics logical rules.
- They are compact.
- They are readable.

How they are implemented:

- Encoded with the abstractions provided by the host language.
- Readability is usually lost in the translation process.
- The effort required for the translation is high.

Semantics of a statement that waits ford a condition or a certain amount of seconds:

$$\frac{\langle t - dt > 0 \rangle \ \Rightarrow \ \texttt{true}}{\langle \texttt{wait}\ t; k\ dt \rangle \ \Rightarrow \ \langle \texttt{wait}\ t - dt; k\ dt \rangle}$$

$$\frac{\langle t - dt > 0 \rangle \ \Rightarrow \ \texttt{false}}{\langle \texttt{wait}\ t; k\ dt \rangle \ \Rightarrow \ \langle k\ dt \rangle}$$

$$\frac{\langle c \rangle \ \Rightarrow \ \texttt{true}}{\langle \texttt{wait}\ c; k\ dt \rangle \ \Rightarrow \ \langle k\ dt \rangle}$$

$$\frac{\langle c \rangle \ \Rightarrow \ \texttt{false}}{\langle \texttt{wait}\ c; k\ dt \rangle \ \Rightarrow \ \langle \texttt{wait}\ c; k\ dt \rangle}$$

STUB Paste the code for wait state machine from Casanova compiler