Lavanda

Daniela da Cruz, Pedro Rangel Henriques

June 23, 2007

In this section we will introduce the case-study: Lavanda language implementation.

1 The problem

Lavanda is a Domain Specific Language (*DSL*) whose main goal is to describe the laundry bags (`Sacos`) that the collection point (`IdPR`) of a big launderette company daily sends to the central building to wash—we call it the ON *ordering note*.
Each bag (`Saco`) is identified by an id number, `num`, and a client name, `IdCli`; its content is composed by one or more items (`Lotes`). Each item (`Lote`) is a subset of laundry of the same type. The type is characterized by: a laundry class, `Classe`, (`corpo`, *body cloth*, or `casa`, *household linen*); a kind of tinged, `Tinto`, (`br`, *white*, or `cor`, *colored*); and a raw-material, `Fio` (`alg`, *cotton*, `la` ,*wool*, or `fib`, *fiber*). For each one item, we register the number of pieces collected.
The (abstract) Context Free Grammar below defines the syntax of the desired language Lavanda. The root of the grammar is `Lavanda`. Terminal symbols are written in lowercase (pseudo-terminals), uppercase (reserved-words), or between apostrophes; the remaining symbols are Non-Terminals. Notice that the concrete grammar is similar, but does not matter for our purpose (semantic specification).

```
p1:       Lavanda →  Cabec  Sacos
p2:       Cabec   →  date  IdPR
p3:       Sacos   →   Saco  '.'
                  |  Sacos  Saco  '.'
p5:       Saco    →   num  IdCli  Lotes
p6:       Lotes   →   Lote
                  |  Lotes Lote
p8:       Lote    →   Tipo  Qt
p9:       Tipo    →   Classe  Tinto  Fio
p10:      IdPR    →   id
p11:      IdCli   →   id
```

```
p12:        Qt       →   num
p13,14:     Classe   →   corpo |  casa
p15,16:     Tinto    →   br    |  cor
p17,18,19:  Fio      →   alg   |  la    |  fib
```

The **problem** consists in *processing an ON*—analyze the laundry bags list, check that there are no two bags with the same id number or the same client name, and compute some numbers. Namely, the demand requires the computation of:

1. the amount of bags received during the day in the collecting point;
2. the total of items per client (the number of items in each bag);
3. the total of pieces per type (for each one of the 12 item types, ranging from '`corpo/br/alg`' until '`casa/cor/fib`');
4. the cost per client (given the price of each laundry type).

In the next section we just will consider the first 3 items above.

2 Attribute Grammar

To solve the problem using an attribute grammar, we take the CFG above as the structural or syntactic basis, and then it is necessary to proceed in 2 steps: (a) choose the attributes (their name, type and class[1]) to associate with each grammar symbol in order to handle all the information needed to compute the required results; (b) write down the attribute evaluation and translation rules, and the contextual conditions associated with each production (grammar rule).

We advocate an incremental approach to AG development. It means that those 2 steps should be applied successively to solve each problem requirement. In our case we execute steps a) and b): four times, to specify the evaluation/translation rules necessary to compute the 4 values demanded; another one, to specify the semantic constraints (i.e. include the contextual conditions). Below, 3 different colors (blue, green and red) are used to distinguish the 3 phases related with the 3 requirements that we will care about.

After that we merge the partial AGs, obtained so far, to produce the solution for the problem.

Table 1 assembles all the attributes chosen, gathering the outcome of step a) for the three phases.

The attribute evaluation and translation rules, necessary to solve sub-problem 1 to 3, are shown below after merging them (notice the colored schema referred above).

```
p1:         Lavanda.nSacos  =  Sacos.nSacos;
            println( Lavanda.nSacos );
```

[1] An attribute should be either *inherited*, or *synthesized*.

Item	Att-Name	Att-Type	Att-Class	Symbols
1	nSacos	int	syn	Lavanda, Sacos
2	nLotes	int	syn	Saco, Lotes
3	inEnv	HashTable	inh	Saco, Lotes, Lote
	outEnv	HashTable	syn	Lavanda, Sacos, Saco, Lotes, Lote
	name	String	syn	Tipo, Classe, Tinto, Fio

Table 1: Attributes used in Lavanda AG

```
            println( Sacos.outEnv )

p3:         Sacos.nSacos   =    1;
            Saco.inEnv  =  Sacos.inEnv;
            Sacos.outEnv  =  Saco.outEnv

p4:         Sacos0.nSacos  =    Sacos1.nSacos + 1
            Saco.inEnv  =  Sacos1.outEnv;
            Sacos1.inEnv  =  Sacos0.inEnv;
            Sacos0.outEnv  =  Saco.outEnv

p5:         Saco.nLotes   =    Lotes.nLotes;
            println( Saco.nLotes );
            Lotes.inEnv = Saco.inEnv;
            Saco.outEnv = Lotes.outEnv

p6:         Lotes.nLotes   =    1
            Lote.inEnv = Lotes.inEnv;
            Lotes.outEnv = Lote.outEnv

p7:         Lotes0.nLotes  =    Lotes1.nLotes + 1;
            Lote.inEnv = Lotes1.outEnv;
            Lotes1.inEnv = Lotes0.inEnv;
            Lotes0.outEnv = Lote.outEnv

p8:         Lote.outEnv = updateTable(Lote.inEnv,Tipo.name, num)

p9:         Tipo.name = Classe.name ++ Tinto.name ++ Fio.name

p13:        Classe.name = "corpo"
p14:        Classe.name = "casa"
p15,...,19:........
```

2.1 Source-Text: a small example

We show below the source-text that could be submitted for processing to the tool under study:

```
10-11-2007 Carrefour
           1 ClientA (corpo-cor-la 1 , casa-cor-alg 2)
           2 Clientb (corpo-cor-fib 10 )
```