Feature grammar systems. Incremental maintenance of indexes to digital media warehouses

Windhouwer, M.A.

Appendix A

The Feature Grammar Language

This appendix contains the feature grammar language in an EBNF notation, *i. e.* the notation as used by the W3C for the XML specification [W3C00].

```
1  #character classes
2  digit             ::= [0-9]
3  exponent          ::= [Ee][-+]?{D}+
4  letter            ::= [_a-zA-Z]
5  any               ::= [#x0-#xFFFE]

6  #literals or constants
7  float             ::= '-'? digit+ '.' digit+ exponent?
8  integer           ::= '-'? digit+
9  unsigned-integer::= digit+
10 string            ::= '"' any* '"'
11 constant          ::= float | integer | string

12 #a symbol
13 symbol            ::= letter ( letter | digit )*

14 #a scope
15 scope             ::= letter ( letter | digit )*

16 #the prefix puts an symbol in a scope, this scope
17 #may refer to a feature grammar, an ADT module or
18 #a detector plugin
19 prefix            ::= scope "::"

20 #simplified XPath expression
```

```
21 xpath            ::= absolute | relative
22 absolute         ::= '/' relative? | "//" relative
23 relative         ::= step ( '/' step | "//" step )*
24 step             ::= axis? ( ( symbol | '*' ) | dereference )
25                    | abbreviation
26 axis             ::= "self::" | "parent::" | "child::"
27                    | "ancestor::" | "ancestor-or-self::"
28                    | "preceding::" | "preceding-sibling::"
29                    | "descendant::" | "descendant-or-self::"
30                    | "following::" | "following-sibling::"
31 abbreviation     ::= '.' | ".."
32 #a feature grammar specific addition
33 dereference      ::= '&' symbol

34 #a list of detector parameters, if no axis is specified for
35 #the first step it defaults to preceding::
36 detector-params ::= '(' ( detector-param ( ','
37                          detector-param )* ) ')'
38 detector-param  ::= constant | xpath

39 #even more simplified XPath expression
40 s-path           ::= s-absolute | s-relative
41 s-absolute       ::= '/' s-relative? | "//" s-relative
42 s-relative       ::= s-step ( '/' s-step | "//" s-step )*
43 s-step           ::= s-axis? ( symbol | '*' )
44 s-axis           ::= "self::" | "child::"
45                    | "descendant::" | "descendant-or-self::"

46 #a list of start symbol parameters, if no axis is specified
47 #for the first step it defaults to the standard child::
48 start-params     ::= '(' ( start-param ( ','
49                          start-param )* ) ')'
50 start-param      ::= s-path

51 #collection type and bounds specification for symbols on the
52 #right-handside of a rule
53 bounds           ::= list | set | tuple
54 list             ::= '[' range ']' | range
55 set              ::= '{' range '}'
56 tuple            ::= '<' int-range '>'
57 range            ::= wild-range | int-range
58 int-range        ::= unsigned-integer
59                      ( ':' unsigned-integer )?
60 wild-range       ::= '*' | '+' | '?'

61 #the feature grammar language, i.e. the start symbol of this
```

```
#EBNF
feature-grammar ::= module-decl decl*
decl            ::= use-decl | start-decl
                  | poll-decl | atom-decl
                  | detector-decl | classifier-decl
                  | version-decl | rule-decl

#module declarations
module-decl     ::= "%module" scope ';'
use-decl        ::= "%use" scope ( ',' scope )* ';'

#start declaration
start-decl      ::= "%start" symbol start-params ';'

#poll declaration
poll-decl       ::= "%detector" symbol ".poll" start-params ';'

#atom and atom rule declarations
atom-decl       ::= "%atom" prefix? symbol
                    ( ( ( symbol ( ',' symbol )* )?
                    | '{' any+ '}' ) )? ';'

#detector declarations
detector-decl   ::= "%detector" prefix? symbol detector-params ';'
detector-decl   ::= "%detector" prefix? symbol
                    '[' any+ ']' ';'

#classifier declaration
classifier-decl ::= "%classifier" prefix? symbol detector-params ';'

#version declaration
version-decl    ::= "%version" symbol
                    unsigned-integer '.'
                    unsigned-integer '.'
                    unsigned-integer ';'

#rule declaration
rule-decl       ::= rhs ':' lhs ';'
rhs             ::= prefix? symbol
lhs             ::= ( '&'? prefix? symbol bounds? | constant )+
lhs             ::= '(' lhs ')'
lhs             ::= lhs '|' lhs
```