Using Perl 6 in Real Life $work

The European Perl Conference,

August 2018, Glasgow

Laurent Rosenfeld,

Paris Perl Mongers

Current Work Situation

- I am working with *Admin*, a large application for managing customers of a tier-1 telecommunication operator
  - Application Admin runs under OpenVMS with a RMS database
  - It consists primarily of 3,290+ programs in the proprietary G programming language (roughly equivalent to PL-SQL under Oracle)
  - The glue language for managing large processes is DCL, more or less the equivalent to the shell under VMS : there are 750 DCL or « .COM » scripts
  - Our team is using a few dozen Perl programs under VMS, but we're stuck with Perl 5.8
  - No chance to use Perl 6 on such an old system as VMS

The New Upgrade Project

- The plans for upgrading Admin are now as follows :
  - Move the OS to Linux (RHEL)
  - Migrate the database to a Cassandra ring (BigData, NoSQL)
  - Translate the G programs to Java
  - Replace the DCL (« .COM ») scripts with shell scripts

- In this context, I was able to use Perl 6 grammars for two subprojects:
  - A proof-of-concept translation of a small subset of the G syntax into Java
  - A detailed analysis of the DCL scripts for the purpose of automatizing documentation

Translating G programs to Java

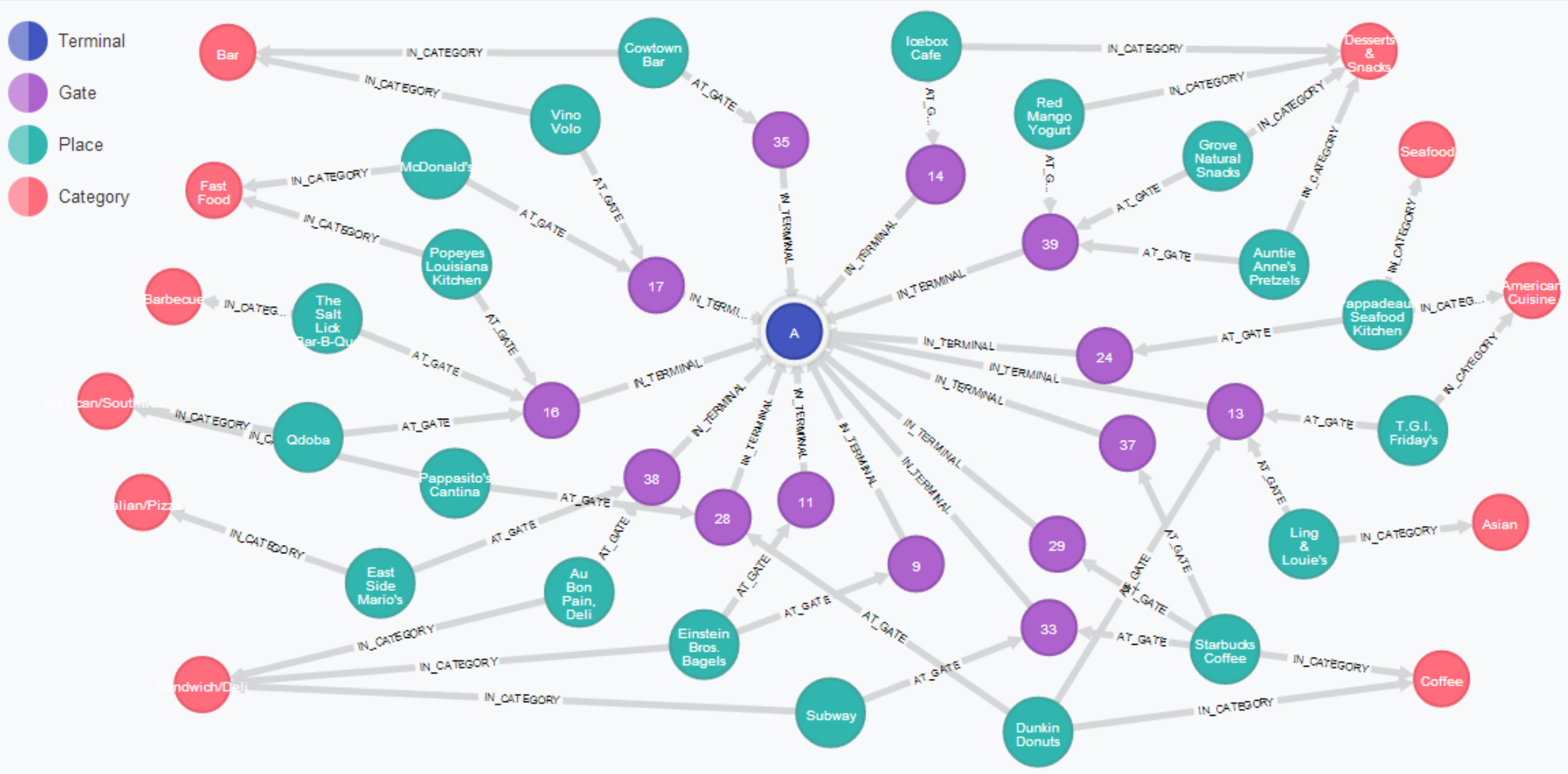
- We used a Perl 6 grammar to generate Java code from G programs
  - This was a POC (in the context of a Request for proposals)
  - Only a small subset of the G syntax
  - Generated Java code that would compile cleanly and even run (provided the right environment setup)
- Our company's bid was not retained (for other reasons)
- However, one of the shortlisted bidders decided to ally with us to eventually win the tender
- So our company is their subcontractor and we're back into the project

Building a complete documentation database

- We wanted to have a complete view of the object relations in Admin
  - Belonging to: field A belongs to tables X, Y and Z
  - Containing: table X contains fieds A, B and D
  - Launching: program A launches programs X and Z
  - Launched by: program X is launched by programs A and F
  - Reading data: program Y reads field A in table Z
  - Writing data: program Y writes field B in table X
  - Is affected by: field B of table X is written by programs X and Y
  - Etc.
- We can dispaly relationships with a Graph database such as Neo4J

Neo4J: expressing relations between nodes

```
CYPHER MATCH p = (:Category)<--(:Place)-[*]->(:Terminal {name:'A'}) RETURN p
```



Populating the graph database

- For fields, tables, and many other objects :
  - It is quite easy to extract the information from the Admin database data dictionary or metadata and other sources, and feed it to the Neo4G graph database
  - Mostly done in Perl 5 with lots of regexes
- For the G programs:
  - We have already developed a grammar for parsing G programs and generating Java code.
  - It is relatively easy to also generate Cypher code for populating Neo4J
- We didn’t have the equivalent for DCL scripts
  - That’s where I used a Perl 6 grammar to parse DCL scripts and generate CSV output and Cypher code to populate Neo4J

A small Glimpse on DCL Syntax

- This is an example DCL script:

```
$ w = "write sys$output"
$ w "Parameter: ''P1'"
$ LOOP:
$       sh queue /all/by *$BATCH/output=sys$login:'P1'CHPREX0201.RECH
$       search sys$login:'P1'CHPREX0201.RECH "CHPREX0201_''P1'"/output=nl:
$       st = $severity
$       if st .eq. 1
$               then
$               del/nolog sys$login:'P1'CHPREX0201.RECH;*
$               wait 00:00:10
$               goto loop
$       endif
$ ENDLOOP:
$ del/nolog sys$login:'P1'CHPREX0201.RECH;*
$ @ECD_COMMANDS:NL_COMP_S 'P1'
```

Dealing with multiline input

- In DCL, new lines are statement separators
  - But some statements can be spread over several lines
  - A line ending with a dash ("-") continues over the next line

```
grammar VMS-grammar {
    rule TOP { [<multi-line> || <line>] + %% \v+ }
    token multi-line { <line-continued>+ <line>  }
    token line-continued {
        ^^ [ <comment-line> || <line-with-keyword> ||
        <line-without-keyword>  ] '-' \h* $$
    }
```

What is a line in DCL ?

- For our purposes, a line can be one of many things :

```
token line {
    || <empty-line>
    || <description-line>
    || <comment-line>
    || <line-with-keyword>
    || <line-without-keyword>
 }
 regex empty-line {
    | ^^ \h*?  $$
    | ^^ '$' \h*? $$          # lines usually start with $
 }
```

Description lines, comment lines

- We need to extract program descriptions and discard comments.
  - Description lines can have two formats:

```
token description-line {
    | ^^ '$' \h* Description \h* '='+ \h* '"'
        (<-["]>+) '"' \N* $$
    | ^^ '$' \h* '!' \N*? :i description \h+ ':' \h+ (\N+) $$
}
```

  - Comment lines start with an exclamation mark :

```
token comment-line { ^^ '$' \s* '!' \N* $$ }
```

VMS Commands

- Lines with various VMS commands:

```
token line-with-keyword {
   ^^ '$' \s* \N*? [ <launch-prg> | <vms_cmd> ] \N* $$
}
regex line-without-keyword {
    ^^ '$' \s* \N*? <assignment>? \N*? $$
}
rule vms_cmd { <vms_cmd_1_arg> | <vms_cmd_2_arg> }
token vms_cmd_1_arg { :i
    | create \h+ "/log"? \h+ ['\fdl=k_fdl:'
        \w+ \.fdl]? <capt-string>
    | delete \h+ "/log"? \h+ <quote>? <capt-string>
}
```

VMS Commands (2)

```
token vms_command_2_arg { :i
    [ append | sort | convert | merge ] \h+
        <option>+ \h* <ident1=capt-string>
        \h+ <ident2=capt-string>
}
token option { \s* "/" \w+['=' \S+]? }
token launch-program { :i
    | (perform) \N+? <quote> k_programs \w*? ':'
        <ident> [\.dmc]? <quote> <dmc_params>?
    | (submit) \N+? k_commands \w*? \: <ident> [\.com]?
    | (\@ k_commands) \: <ident> ".com"?
}
```

VMS Commands (3)

```
token dmc_params { '(' <-[)]>+ ')' }
token assignment {
   | <id> \s* ':'? '='+ \s* 'f$sear' [ch]* '(' <value> ')'
   | <id> \s* ':'? '='+ \s* <value>
}
token id { \w+ }
token capt-string {
    | \w <[ \s \w ]>+
    | \w+ ':' \w+ [\. \w+]?
    | \w+     }
token value { <quote>* <capt-string> <quote>* }
token quote { '"' | "'" }
```

Excerpts from the Actions

```
class VMSactions {
  method assignment($/) { %*var{~$/<ident>} = ~$/<value><capt-string>; }
  method description-line($/) { $*prog-description = $0; }
  method vms_command_1_arg($/) {
      push @*result, {node-type => "logical", rel-type => "node_logic",
          dest => $<capt-string>, dest-type => "LOGICAL"};
      push @*result, {node-type => "relation", rel-type => "rel_logic",
          dest => $<capt-string>, dest-type => "LOGICAL",
          rel-name => "Access" };
      }
  method vms_command_2_arg($/) {
      push @*result, {node-type => "logical", rel-type => "node_logic",
          dest => $<ident1>, dest-type => "LOGICAL"};
      push @*result, {node-type => "logical", rel-type => "node_logic",
          dest => $<ident2>, dest-type => "LOGICAL"};
  }
```

Calling the Grammar for one DCL Script

```
sub process_one_file ($vms-file) {
    my ( @*result, @*assignments, %*var, $*prog-description );
    my $root-file-name = $0 if $vms-file ~~ / :i (\w+\.com$)/;
    my $match = VMS-grammar.parsefile($vms-file,
        :enc('iso-8859-1'), :actions(VMSactions));
    print-output($root-file-name) if $match;
    say "matched:\t$root-file-name" if $match;
}
sub MAIN (Str $param) {      # $param = a DCL file or a directory
    # ... Looping on the .COM files in a directory
        process_one_file ($filename)
}
```

Note on Performance

- Is Perl 6 fast enough for that?
- Parsing 745+ DCL scripts, representing 5 MB and 106 k lines of code
- Runs in less that 30 seconds on this PC
- We can safely run the whole process, including this Perl 6 program, for every new release of Admin (a full release every second month)

Conclusion

- I was happy to be able to use Perl 6 on a real work project.
- Using a real grammar really made sense for that part of the project and this was very successful
- Thank you for listening.
- Do you have any questions?

  - These slides are available on Github (https://github.com/LaurentRosenfeld
  - They can be used under the terms of the Creative Common Attribution ShareAlike License (CC-BY-SA)