Research results into the efficacy and viability of ANTLR as a server side language parsing technology by Glenn Engstrand

The ability to parse strings in a known and non-ambiguous language to determine context sensitive knowledge is a powerful concept both in theory and practice. In computer science, this is known as compiler theory. I remember getting to play with YACC (Yet Another Compiler Compiler) after taking two courses in compiler theory back in my college days. The modern day equivalent of YACC is ANTLR which is ANother Tool for Language Recognition. The current version of ANTLR is 3 which has some pretty interesting features including the ability to use ANTLR to write filters, to have a look-ahead of indefinite size (more on this later) and to use ANTLR to embed language parsers in a wide variety of different programming languages and platforms including Java, C#, and Python.

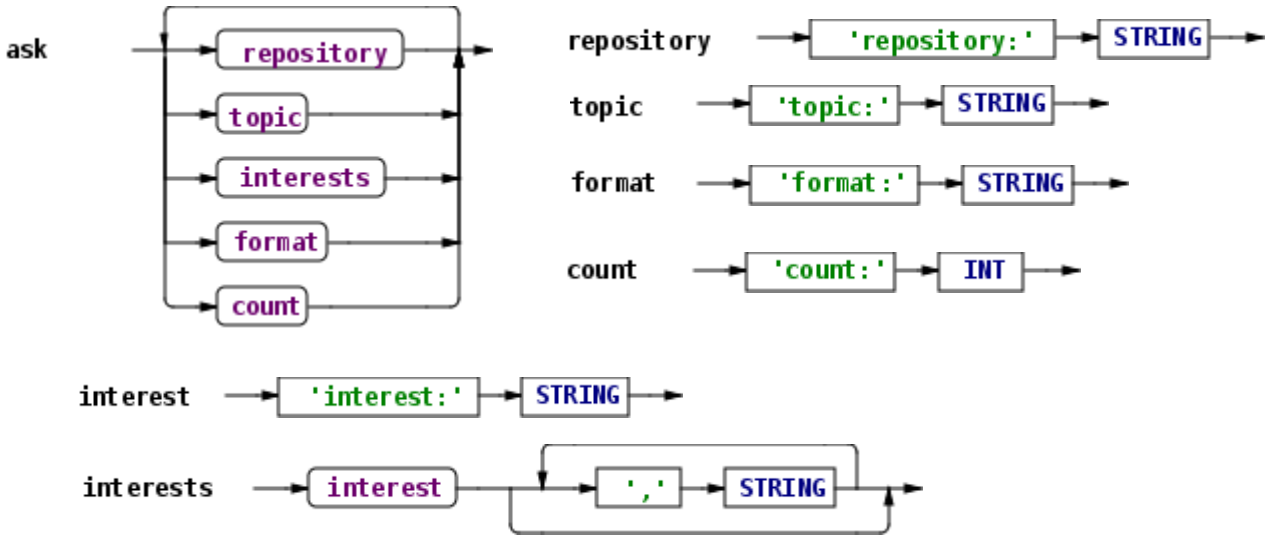
I have used ANTLR many times in the past but mostly as a way to embed programming and configuration language capability in a command line tool for support or configuration purposes. What I was interested in discovering was how effective ANTLR could be in server side technology. Is ANTLR both scalable and per formant enough for the embedding of domain specific language capability in server technology?

Parsing a string intended to be written in a particular language is kind of like finding your way in a maze. There are a lot of choices that have to be made and possible back tracking to determine whether or not this is a valid string. The language is specified in something that looks a lot like EBNF or Extended Backus-Naur Format. YACC is a LALR(1) parser which means that it looks ahead left to right and only one level deep. Previous versions of ANTLR were an LL(k) parser which allowed it to look head k levels deep (i.e. a number provided as a part of the grammar specification). The latest version of ANTLR is an LL(*) parser which means that it can look ahead any number of steps. This makes for specifying your EBNF style grammar in a much more human friendly and readable way.

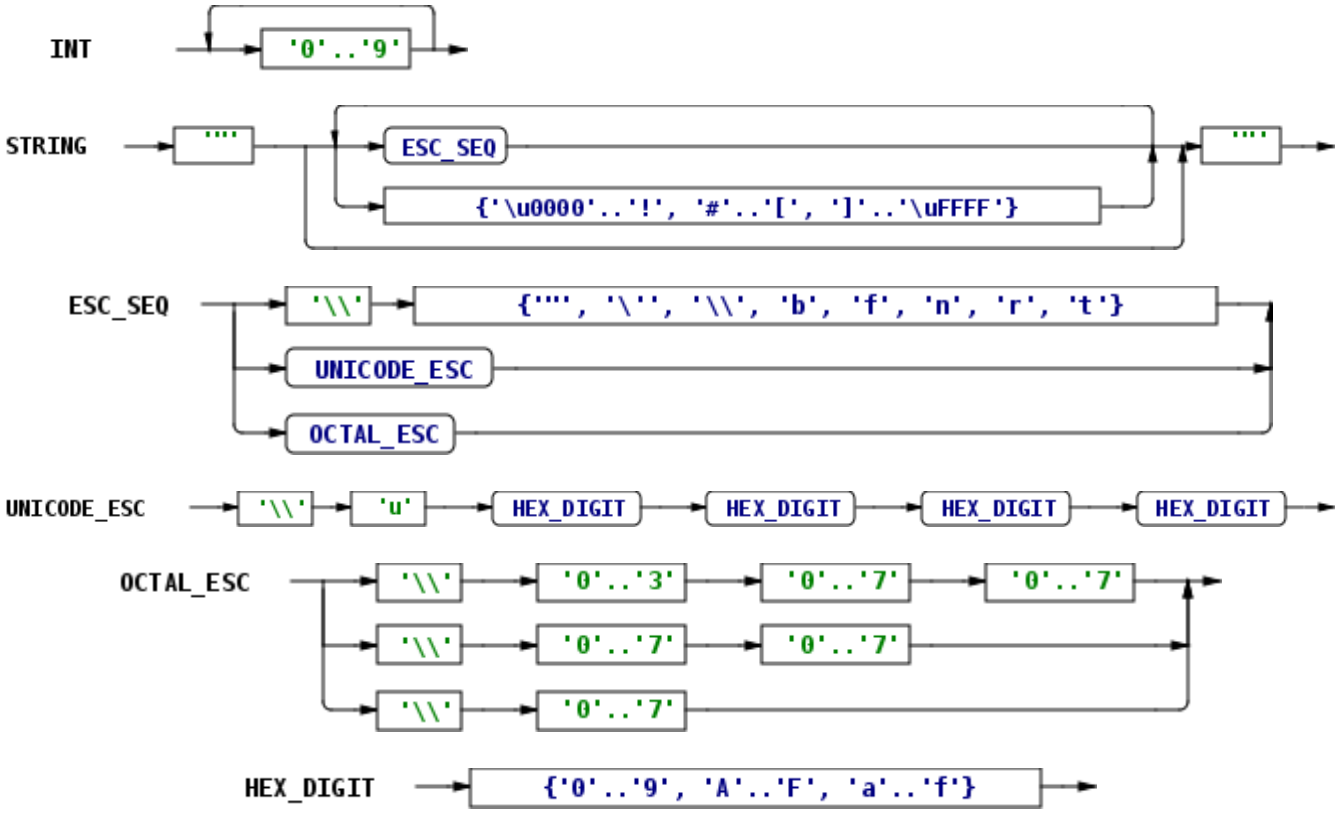
What do you do with a string once it has been parsed to determine whether or not it is compliant to the intended language specification? With ANTLR, there are three things that you can do with it. You can embed your own commands in the parser code. You can have the parser construct an Abstract Symbol Tree that can be traversed to get meaning out of it. You can instruct the grammar to output a string using a technology called string template.

What I did was explore the use of ANTLR to parse the query string of web requests by invoking the ANTLR technology in a servlet hosted by the Tomcat application container.

Here is the language specification that was used in this experiment.



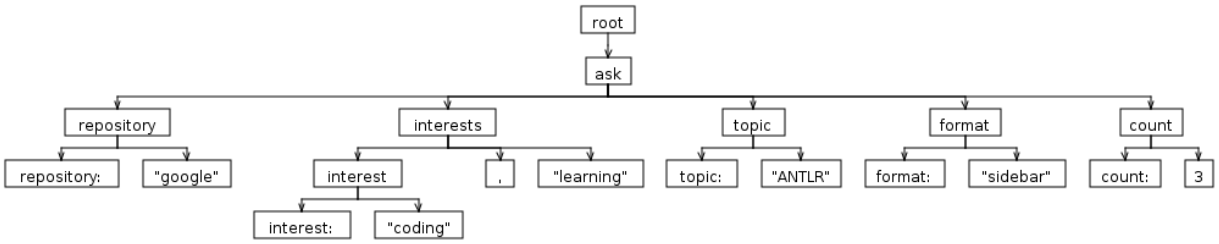
Parsers like ANTLR actually split up the job into two passes, a lexical analysis pass and the actual parsing pass itself. Here is the grammar specification for the lexical analysis phase of the parsing.



There must also be a way to tell ANTLR what to do with white space which is the part of the language specification that makes strings written in this language to be human friendly to read without impacting

the semantic meaning of the expression itself.

Here is an illustrative sample of a parse tree using this grammar.



Like I said before, you can embed your commands into a parser specification or you can write code to traverse the AST or Abstract Symbol Tree after parsing has completed. Here is the grammar specification for the above language with the commands embedded within it.

```
grammar NewsRequest;

options {
language=Java;
}
@header {
package com.dynamicalsoftware.news.parser;
import java.io.PrintWriter;
}
@lexer::header {
package com.dynamicalsoftware.news.parser;
}
@members {
public PrintWriter writer = null;
}

ask  :
     (
     repository
     |
     topic
     |
     interests
     |
```

```
     format
     |
     count
     )+
     ;
repository     :
     'repository:' STRING
     { writer.write("<tr><td>repository</td><td>" + $STRING.text +
"</td></tr>"); }
     ;
topic     :
     'topic:' STRING
     { writer.write("<tr><td>topic to cover</td><td>" + $STRING.text
+ "</td></tr>"); }
     ;
interests :
     interest ( ',' STRING
     { writer.write("<tr><td>area of interest</td><td>" +
$STRING.text + "</td></tr>"); }
      )*
     ;
interest  :
     'interest:' STRING
     { writer.write("<tr><td>area of interest</td><td>" +
$STRING.text + "</td></tr>"); }
     ;
format    :
     'format:' STRING
     { writer.write("<tr><td>format of response</td><td>" +
$STRING.text + "</td></tr>"); }
     ;
count     :
     'count:' INT
     { writer.write("<tr><td>max number of items to return</td><td>"
+ $INT.text + "</td></tr>"); }
     ;
INT :     '0'..'9'+
    ;

WS  :   ( ' '
        | '\t'
        | '\r'
        | '\n'
        )+ {$channel=HIDDEN;}
```

```
    ;

STRING
    :  '"' ( ESC_SEQ | ~('\\'|'"') )* '"'
    ;

fragment
HEX_DIGIT : ('0'..'9'|'a'..'f'|'A'..'F') ;

fragment
ESC_SEQ
    :   '\\' ('b'|'t'|'n'|'f'|'r'|'\"'|'\''|'\\')
    |   UNICODE_ESC
    |   OCTAL_ESC
    ;

fragment
OCTAL_ESC
    :   '\\' ('0'..'3') ('0'..'7') ('0'..'7')
    |   '\\' ('0'..'7') ('0'..'7')
    |   '\\' ('0'..'7')
    ;

fragment
UNICODE_ESC
    :   '\\' 'u' HEX_DIGIT HEX_DIGIT HEX_DIGIT HEX_DIGIT
    ;
```

This is how you invoke this parser.

```
     private void parse(String cmd, PrintWriter writer) throws
RecognitionException {
           ANTLRStringStream input = new ANTLRStringStream(cmd);
           NewsRequestLexer lex = new NewsRequestLexer(input);
           CommonTokenStream tokens = new CommonTokenStream(lex);
           NewsRequestParser parser = new NewsRequestParser(tokens);
           parser.writer = writer;
           parser.ask();
     }
```

Here is the specification for parsing strings of the same language only this time generating an AST.

```
grammar NewsRequestAST;

options {
output=AST;
language=Java;
}

@header {
package com.dynamicalsoftware.news.parser;
}
@lexer::header {
package com.dynamicalsoftware.news.parser;
}

ask  :
     (
     repository
     |
     topic
     |
     interests
     |
     format
     |
     count
     )+
     ;
repository     :
     'repository:' STRING -> ^(  'repository:' STRING )
     ;
topic     :
     'topic:' STRING -> ^( 'topic:' STRING )
     ;
interests :
     'interest:' STRING ( ',' STRING )* -> ^( 'interest:' STRING+ )
     ;
format    :
     'format:' STRING -> ^( 'format:' STRING )
     ;
count     :
     'count:' INT -> ^( 'count:' INT )
     ;
```

```
INT :     '0'..'9'+
    ;

WS  :   ( ' '
        | '\t'
        | '\r'
        | '\n'
        )+ {$channel=HIDDEN;}
    ;

STRING
    :  '"' ( ESC_SEQ | ~('\\'|'"') )* '"'
    ;

fragment
HEX_DIGIT : ('0'..'9'|'a'..'f'|'A'..'F') ;

fragment
ESC_SEQ
    :   '\\' ('b'|'t'|'n'|'f'|'r'|'\"'|'\''|'\\')
    |   UNICODE_ESC
    |   OCTAL_ESC
    ;

fragment
OCTAL_ESC
    :   '\\' ('0'..'3') ('0'..'7') ('0'..'7')
    |   '\\' ('0'..'7') ('0'..'7')
    |   '\\' ('0'..'7')
    ;

fragment
UNICODE_ESC
    :   '\\' 'u' HEX_DIGIT HEX_DIGIT HEX_DIGIT HEX_DIGIT
    ;
```

This is how you invoke this parser.

```
      private void parseAST(String cmd, PrintWriter writer) throws
RecognitionException {
            ANTLRStringStream input = new ANTLRStringStream(cmd);
            NewsRequestASTLexer lex = new NewsRequestASTLexer(input);
            CommonTokenStream tokens = new CommonTokenStream(lex);
            NewsRequestASTParser parser = new NewsRequestASTParser(tokens);
```

```
            Tree t = (Tree)parser.ask().getTree();
            for (int i=0; i<t.getChildCount(); i++) {
                  Tree child = t.getChild(i);
                  String action = child.getText();
                  for (int j=0; j<child.getChildCount(); j++) {
                        writer.write("<tr><td>" + action + "</td><td>" +
child.getChild(j).getText() + "</td></tr>");
                  }
            }
      }
```

I was not able to successfully use string template in the Tomcat 6 environment.

Invoking these methods in an HttpServlet was cake.

```
      protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
            PrintWriter writer = response.getWriter();
            String qs = request.getQueryString();
            qs = URLDecoder.decode(qs, Charset.defaultCharset().name());
            String[] nv = qs.split("&");
            String cmd = "";
            String action = "embedded";
            for (String nvp : nv) {
                  String[] eqp = nvp.split("=");
                  if (eqp.length == 2) {
                        if (eqp[0].equalsIgnoreCase("a")) {
                              action = eqp[1];
                        } else if (eqp[0].equalsIgnoreCase("c")) {
                              cmd = eqp[1];
                        }
                  }
            }
            writer.write("<html><head><title>test antlr
servlet</title></head><body><h1>Parse Results</h1><p>action = " + action +
"</p><p>cmd = " + cmd + "</p><table>");
            try {
                  Date before = new Date();
                  if (action.equalsIgnoreCase("embedded")) {
                        parse(cmd, writer);
                  } else if (action.equalsIgnoreCase("AST")) {
                        parseAST(cmd, writer);
                  }
                  Date after = new Date();
                  writer.write("<tr><td>milliseconds</td<td id=\"duration\">" + new
Long(after.getTime() - before.getTime()).toString() + "</td></tr>");
            } catch (RecognitionException e) {
                  writer.write("<tr><td>invalid</td><td id=\"invalid\">" +
e.getLocalizedMessage() + "</td></tr>");
            } catch (Exception e) {
                  writer.write("<tr><td>error</td><td id=\"error\">" +
```

```
e.getLocalizedMessage() + "</td></tr>");
            } finally {
                  writer.write("</table></body></html>");
            }
      }
```

Here is what the resulting web page looks like.

Parse Results

action = embedded

cmd = repository: "yahoo" interest: "shopping" format: "activity" topic: "cell phone" count: 21

repository	"yahoo"
area of interest	"shopping"
format of response	"activity"
topic to cover	"cell phone"
max number of items to return	21
milliseconds	1

In order to stress test ANTLR in a servlet environment, I repeatedly reloaded this, and other, pages that invoke the servlet and noted the reported latency (in milliseconds). At the same time, I had set up Tsung to also perform similar testing.

```
?xml version="1.0"?>
<!DOCTYPE tsung SYSTEM "/usr/share/tsung/tsung-1.0.dtd" [] >
<tsung loglevel="info">
  <clients>
    <client host="glenn-laptop" use_controller_vm="true"/>
  </clients>
  <servers>
    <server host="localhost" port="8080" type="tcp"></server>
  </servers>
  <load>
    <arrivalphase phase="1" duration="60" unit="minute">
      <users interarrival="1" unit="second"></users>
    </arrivalphase>
  </load>
  <sessions>
    <session name="embedded" probability="50" type="ts_http">
      <request>
      <http url="/NewsWidget/ask?a=embedded&amp;c=repository:%20%22google
%22%20interest:%20%22coding%22%20,%20%22learning%22%20format:%20%22sidebar
%22%20topic:%20%22Google%20Web%20Toolkit%22%20count:%203" method="GET"
version="1.1"></http>
      </request>
    </session>
    <session name="AST" probability="50" type="ts_http">
```

```
      <request>
      <http url="/NewsWidget/ask?a=AST&amp;c=repository:%20%22yahoo%22%20interest:
%20%22shopping%22%20format:%20%22activity%22%20topic:%20%22smart%20phones
%22%20count:%2012" method="GET" version="1.1"></http>
      </request>
    </session>  </sessions>
</tsung>
```

Running this test stressed the servlet upwards of two times per second for an hour. ANTLR continued to perform at sub millisecond speeds during the entire experiment. Also, there were no errors reported during this operation. From these admittedly preliminary findings, I must conclude ANTLR to be a viable and stable method of server side parsing.

References

LALR Parsing
http://dragonbook.stanford.edu/lecture-notes/Stanford-CS143/11-LALR-Parsing.pdf

ANTLR Parser Generator
http://antlr.org/

The Definitive ANTLR Reference: Building Domain-Specific Languages
http://www.pragprog.com/titles/tpantlr/the-definitive-antlr-reference

Java Servlet Technology
http://www.oracle.com/technetwork/java/servlet-138661.html

Extended Backus-Naur Format
http://www.cs.cmu.edu/~pattis/misc/ebnf.pdf

Apache Tomcat
http://tomcat.apache.org/

Tsung is an open-source multi-protocol distributed load testing tool.
http://tsung.erlang-projects.org/