Prácticas docentes en Teoría de Autómatas y Lenguajes Formales (TALF) con la herramienta ANTLR/ANTLRWorks

Eric Jeltsch F.
Departamento de Matemáticas.
Universidad de La Serena
La Serena, Chile
e-mail: ejeltsch@userena.cl

Abstract—Se presentan algunas experiencias docentes realizadas en la asignatura de Teoría de Autómatas y Lenguajes Formales con las herramientas ANTLR y su IDE ANTLRWorks. En el transcurso de las actividades se comprobó la potencialidad y efectividad de ellas en el contexto de analizadores léxicográficos y sintácticos. Las experiencias se basan en una variedad de actividades que hacen que la incorporación de este tipo de herramientas sea recomendable, oportuna y factible de incluir en un curso de gramáticas y lenguajes formales para estudiantes de Ingeniería en Computación.

Keywords; prácticas docentes; procesadores de lenguajes; antlr; lenguajes formales.

I. INTRODUCCION

La presente propuesta se inserta dentro de las actividades regulares de la sección de Laboratorio en la asignatura de TALF (Teoría de Autómatas y Lenguajes Formales) para la carrera de Ingeniería en Computación de la Universidad de La Serena (ULS). El objetivo de los Laboratorios es que el alumno clarifique, desarrolle, aplique e implemente soluciones a problemas que le son propios, en el ámbito de la asignatura. Lo novedoso de la propuesta es que en esta versión del curso las actividades deben ser resueltas con ANTLR (ANother Tool for Language Recognition) [5] y el uso del entorno de desarrollo ANTLRWorks [7]. TALF es una asignatura teórico- práctico del 7º Nivel, con una estructura de contenido clásica, así como las herramientas utilizadas en semestres anteriores, tales como Lex/Yacc, Flex/Bison, en el entorno del lenguaje C, y JLex/CUP en el entorno Java. La asignatura posee una jornada de 6 horas (TEL: 4-0-2, esto es 4 horas de Teoría, 0 horas de Ejercicio y 2 horas de Laboratorio). Los prerequisitos de la asignatura son: Programación (Orientada a Objeto, POO) y Diseño y Análisis de Algoritmos (DAA), cuyas actividades de Laboratorio están integramente enfocadas al lenguaje Java, con uso intensivo de las librerías, `Generic`, `Collections` y `Swing`, entre otras. En el mismo 7º Nivel, los alumnos están cursando la asignatura de Ingeniería de Software, Base de Datos y Comunicación de Datos y Redes, todas ellas con un TEL: 4-0-2. Con esto se desea definir el entorno curricular y las expectativas (objetivos) que sustentan la propuesta temática y aplicaciones dentro de TALF. La asignatura está inserta en el cuarto año de la carrera, de un total de cinco (10 Semestres) y es coherente con los referentes universales, esto es, con las recomendaciones de Computing Curricula 2005 de la ACM/IEEE-CS Curriculum Joint Task- Force, en el sentido de que las materias y conocimientos de TALF se encuadran dentro del área de conocimiento de los Lenguajes de Programación (PL), tales como, (PL3. Introducción a la traducción de lenguajes) y (PL8. Sistemas de traducción de lenguajes) [20]. El principal objetivo pedagógico de TALF es dotar al alumno de las competencias básicas para la descripción sistemática de lenguajes formales y el desarrollo sistemático de intérpretes, filtros y transformaciones en los lenguajes formales, siguiendo técnicas estándares que se ponen en práctica en los Laboratorios. La propuesta muestra los fundamentos que la originaron, ámbitos en donde se aplican los lenguajes específicos, junto con una introducción a ANTLR/ANTLRWorks como herramientas de apoyo durante las actividades de Laboratorio. Se muestra además un ejemplo que va mostrando las dificultades o conceptos que deben ser abordados en teoría, para lograr los resultados esperados. Se plantean los objetivos y formas de evaluar las actividades, así como opiniones de los alumnos sobre las mismas, para finalmente en las conclusiones dar algunas pautas de cómo seguir en futuras versiones.

II. FUNDAMENTOS

Los primeros intentos de incluir algunas herramientas computacionales en la asignatura TALF, distintas a las clásicas (Lex/Yacc), se remontan al año 2002, para cuando se realiza un estudio del problema de Inferencia Gramatical para gramáticas y lenguajes formales de grafos, y su posterior aplicación y visualización en TreeBag, [15,16, 17]. Ahora, con la importancia y dinámica de los procesadores de lenguajes se dicta el año 2010 un curso electivo, cuyo objetivo es conocer el ámbito y alcances de los Lenguajes de Dominio Específico (DSL, *Domain Specific Language*). Esta experiencia, sumada a la implementación de la estandarización de protocolos de comunicación del sistema SAT (Suite Aplicaciones Telemedicina) con las especificaciones HL7 [24], dieron origen a tener que construir lenguajes o especificaciones formales en el ámbito de la Telemedicina (e-Health) con especificaciones HL7 [8,

18,19], para entender y participar de la interoperabilidad "semántica" en los sistemas de salud, siendo este último uno de los temas de investigación del autor. La importancia de los DSL radica en que son lenguajes de programación que están diseñados para utilizarse en dominios o problemas específicos, a diferencia de los lenguajes de programación de propósito general (Java, C, C++ u otros). Martin Fowler y Debasish Ghosh hacen una introducción y clasificación muy interesante de los mismos en [22, 23]. En resumen, con las experiencias antes recabadas se pudo confirmar que un usuario familiarizado con el dominio del lenguaje puede obtener mejores y más rápidos resultados utilizando una semántica de dominio conocida, y por otro, al utilizar Java cómo lenguaje contenedor permitió desarrollar aplicaciones en un dominio concreto, además de seguir usando los entornos de desarrollo Eclipse, NetBeans, IntelliJIdea, Maven, ApacheCamel, entre otros. Ahora, ante los desafíos conducentes a la acreditación y actualización de los contenidos de las asignaturas y planes de estudio de la carrera es que se está estudiando la incorporación formal de las gramáticas ANTLR y el IDE (Entorno desarrollo integrado) ANTLRWorks [7], las que esperamos coadyuven en conjunto con las herramientas clásicas a la consecución del objetivo pedagógico de la asignatura TALF.

III. AMBITOS DE APLICACIÓN

Actualmente la literatura ofrece una gran variedad de problemas y ámbitos de aplicación en donde DSL y en particular ANTLR son actores importantes de la solución. Por ejemplo, en Telemedicina la codificación específica de X73PHD, el cual es un estándar en la captura de la información recogida por los dispositivos médicos (*Medical Devices*, MDs, y *Personal Health Devices*, PHDs), ASN.1 que es un lenguaje orientado a la definición de protocolos, sobre el cual se han diseñado los tipos de datos a gestionar, las estructuras de los mensajes y modificación al árbol sintáctico, todo ello a través de un DSL construído con ANTLR [9]. En Bioinformática una serie de parsers han sido desarrollados, entre ellos, (E)BNF para partes de Galaxy ToolConfigs con ANTLR, así como la definición de una gramática en la búsqueda de una cierta línea de mutantes, en la Neurociencia, o en la construcción de parser en la Química [1,2,3,4].

En general, son numerosos los proyectos en donde ANTLR ha intervenido, entre ellos, Apache Camel, Apache Lucene, Groovy, Hibernate, en donde se ha construído parser para un lenguaje apropiado. Por ejemplo, Hibernate usa ANTLR para el parser en el lenguaje de consulta HQL, EJB-QL y Criteria Query. En Drools (Business Logic Integration Platform), uno de los productos más importantes en la plataforma JBoss-SOA es el lenguaje DRL, el cual relaciona reglas de negocio definidas a alto nivel dentro de un motor de procesos. Es decir, una cantidad importante de proyectos han visto lo apropiado de usar los DSL en pequeñas aplicaciones y la importancia que le cabe a ANTLR en este aspecto. Una lista de proyectos que usan ANTLR está habilitado en [5], para mayor información. En el ámbito educativo o formativo existen una serie de trabajos de titulación, estudios comparativos, memorias o tesis doctorales que incorporan ANTLR en el logró de sus objetivos, tal como se muestran en el Primer Congreso Internacional de Software Libre Zacatecas 2011, XI Jornadas de Enseñanza Universitaria de la Informática, entre otros [6,10,13,14,26].

Cabe preguntarse, ¿Y qué pasa en Chile con la incorporación de ANTLR en las asignaturas clásicas de prcesadores de lenguajes?. Cómo una forma simple de detectar la oportunidad/necesidad de incorporar esta herramienta en Chile, es que se le ha dado a Google la consulta genérica "`ANTLR y XX`", siendo `XX` un patrón cercano a la(s) asignatura(s) que la pudiesen relacionar, cuyos resultados se muestran en Tabla nº1.

Tabla nº1

Compiladores	Lenguajes	Autómatas	Blogs
65	96	8	1240

En cambio, en Tabla nº2 se muestra la misma consulta, pero en Inglés y en toda la Web.

Tabla nº2

Compilers	Languages	Automata	Blogs
420.000	551.000	2.680.000	427.000

En base a estos resultados, junto a la cantidad de proyectos e iniciativas ya mencionadas, se hace propicio y oportuno el incorporar estas herramientas, para que así vayan obteniendo las habilidades para participar o insertarse en grandes proyectos de desarrollo. Es así como esta propuesta pretende además, instalar el tema y con ello las eventuales dificultades pedagógicas (por lo extenso y variado de los contenidos teóricos que sustentan la inclusión de estas herramientas en los cursos regulares de lenguajes formales). Una mención especial al Dr. Ricardo Soto, por los intentos de incorporarlo en su asignatura ICI 445 (Universidad Católica de Valparaíso) [25] u otros que pudieron ser omitidos. Sin embargo, en el ámbito académico no se muestra en la Web, al menos en Chile, una forma masiva de incluir este tema en cursos regulares de lenguajes, compiladores o procesadores de lenguajes en donde ANTLR sea objeto de estudio en forma más profunda. Por todo lo anterior se hace necesario, oportuno y factible el promover el estudio de los DSL por una parte, y por otra buscar las herramientas que permitan lograr la consecución de los objetivos, entre ellos, la implementación del parser de algún lenguaje, así como, la implementación de analizadores léxico, sintáctico y semántico, así como transformaciones apropiadas bajo especificaciones ANTLR/ANTLRWorks, las que pueden ser más eficientes y efectivas.

IV. INTRODUCCIÓN A ANTLR/ANTLRWORKS

ANTLR es una herramienta que proporciona todo lo necesario para la construcción de analizadores léxicos, sintácticos, recorrido de árboles de sintaxis abstracta (AST, Abstract Sintax Tree), mecanismos de detección y recuperación de errores [12]. Una ventaja de ANTLR con respecto a otras herramientas clásica es que genera el código de salida en diferentes lenguajes, tales como, Java, Ruby, C++, C# o Python, entre otros. Por otro lado, ANTLRWorks permite construir de forma amigable las gramáticas de entrada, proporcionando representaciones gráficas de las expresiones, diagramas sintácticos, parse tree, AST, módulos de entrada y salida, intérprete y depurador.

Funcionalidades:

a) Editor de gramáticas,
b) Visualización de las reglas gramaticales como diagrama sintáctico,
c) Interpreter, para un prototipado rápido,
d) Debugger, para encontrar errores en la gramática,
e) Visualización del lookaheads (DFA),
f) Refactorización, por ejemplo, para la recursividad por la izquierda,
g) Visualización dinámica del Parse Tree,
h) Visualización dinámica del Abstract Syntax Tree, (AST).

En el desarrollo de las actividades propuestas, se pudo comprobar la utilidad de lo antes mencionado, sobretodo a la hora de identificar los errores cometidos en la especificación, precedencia o ambiguedad.

Algunas definiciones básicas:

-Lexer (Análisis lexicográfico) es el proceso que convierte un stream (cadena) de caracteres a un stream (cadena) de tokens.
-Parser (Análisis Sintáctico) es el proceso que posibilita crear un AST (Abstract Syntax Tree), representación intermedia, a partir de la cadena de tokens.
-Tree Parser – procesar un AST.
-StringTemplate es una "template" (plantilla) Java para generar código fuente, páginas web, emails, o cualquier otro formato de texto, como salida. [21]

Características: ANTLR usa gramáticas especificadas en la forma Extended Backus-Naur (EBNF) las que permiten expresar opciones, elementos repetidos o ciclos. ANTLR soporta gramáticas LL(*). LL(k) parser es un "top-down parser" que parsea desde la izquierda a derecha, construye derivaciones más a la izquierda de la entrada y el "lookahead", con k tokens para cuando debe seleccionar entre reglas alternativas.

Áreas de Aplicación: Existen tres casos primarios de uso para ANTLR

a) Implementar "*validadores*", esto es, generar código en forma automática que valida la entrada, basado en las reglas gramaticales.
b) Implementar "*procesadores*", esto es, generar código que válida y procesa la entrada. Con ello es posible realizar cálculos, actualización de bases de datos, lecturas de archivos de configuración con una estructura de datos apropiada, etc.
c) Implementar "*traductores*", esto es, generar código que válida y traduce una entrada en otro formato, también llamado código intermedio, por ejemplo, bytecode en Java.

Etapas generales en el uso de ANTLR: Un enfoque común es el uso de tres archivos. El primero de ellos, es la gramática `MiGra.g` que genera automáticamente los archivos `MiGraLexer.java`, `MiGraParser.java`, y `MiGra.tokens`. En el ámbito de los AST, existen las herramientas para construirse una gramática de árboles, `tree grammar MiGra`, declarada sobre un archivo `MiGra.g`, la cual procesa un AST y las formas de visitar los nodos. Opcionalmente, se pueden escribir plantillas StringTemplate para producir salidas.

En la Figura 1 se observa como ANTLR actúa como intermediario entre la Teoría y la Práctica, permitiendo así, la generación de Lexer, Parsers y Tree Parsers, que automatizan la construcción de *"validadores"*, *"procesadores"* y *"traductores"*.

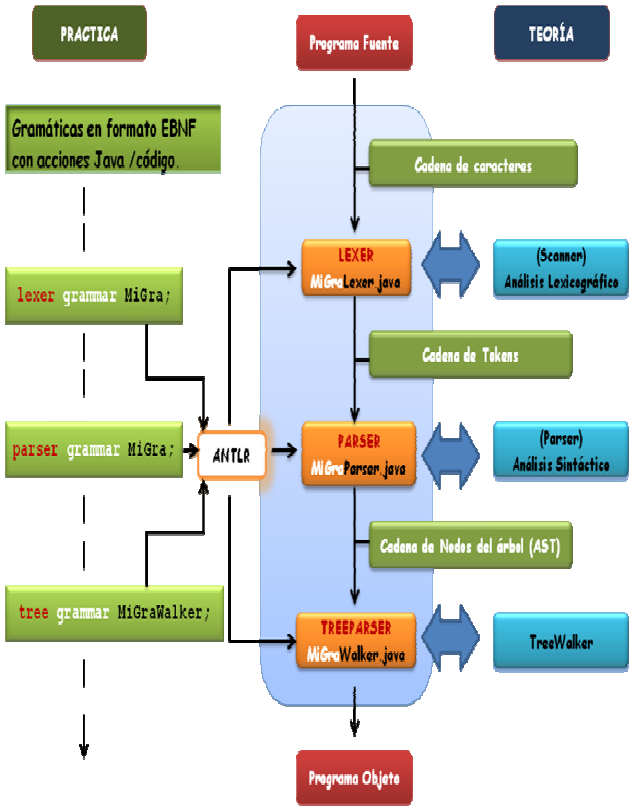

Figura 1. ANTLR, un nexo entre la Teoría y la Práctica.

Un ejemplo

A continuación se muestra un ejemplo para recrear los conceptos de "*validador*" y "*procesador*". Al considerar la expresión aritmética 9+5*2, se propone la gramática de contexto libre `G ={{Expr}; {Number, "+", "-", "*", "/", "(", ")" }, P, Expr)`,donde las producciones o reglas que forman parte del conjunto P son:

```
Expr :: Expr  "+" Expr
     | Expr "-" Expr
     | Expr "*" Expr
     | Expr "/" Expr
     | "(" Expr ")"
     | Number.
```

Claramente la propuesta es ambigua, ya que posee dos interpretaciones (dos árboles sintácticos diferentes), (9+5)*2 o bien 9+(5*2). Volviendo a nuestro ejemplo, vemos que la primera producción debería corresponder a dos no-terminales, `Expr, Number` y un no-terminal `Factor`, que hacen posible generar unidades básicas en las expresiones, según cierta prioridad, o asociatividad. Se muestra a continuación una producción básica para representar una expresión con estas condiciones.

```
Factor :: "(" Expr ")"
       | Number
       ;
```

Ahora, si consideramos los operadores binarios "*" y "/" que tienen mayor precedencia y se asocian por la izquierda, se tienen las producciones.

```
Product :: Product "*" Factor
        | Product "/" Factor
        | Factor
        ;
```

Análogamente, consideramos los operadores binarios "+" y "-", obteniéndose las producciones.

```
Expr :: Expr "+" Product
     | Expr "-" Product
     | Product
     ;
```

Finalmente la gramática en la forma BNF resultante es:

```
Expr :: Expr "+" Product
     | Expr "-" Product
     | Product
     ;
Product :: Product "*" Factor
        | Product "/" Factor
        | Factor
        ;
Factor :: "(" Expr ")"
       | Number
       ;
```

La que según vemos es una gramática recursiva izquierda para expresiones aritméticas, dejándose ver uno de los mayores problemas, ya que ANTLR no soporta éste tipo de gramáticas, de manera que se debe transformar y eliminarla, tal como se muestra a continuación.

```
Expr :: Product ExprRest
ExprRest :: "+" Product ExprRest
         | "-" Product ExprRest
         | "λ"
         ;
Product :: Factor ProductRest
         ;
ProductRest :: "*" Factor ProductRest
            | "/" Factor ProductRest
            | "λ"
            ;
Factor :: "(" Expr ")"
       | Number
       ;
```

Para finalmente obtener una nueva gramática no-recursiva izquierda para expresiones aritméticas en notación EBNF, llamémosle, `expr`.

```
Expr :: Product (( "+" | "-" ) Product)*;
Producto :: Factor (( "*" | "/" ) Factor)*;
Factor :: "(" Expr ")"
| Number
;
```

Como una muestra de la dinámica de las actividades encomendadas en los Laboratorios es que se presenta una extensión de la gramática anterior. En éste caso, se incluyen las reglas Exponencial(`exp`) y Logaritmo(`log`),

```
Expr:: Product(( "+" | "-" )Product)*;
Product:: Factor(( "*" | "/" )Factor)*;
Factor:: "(" Expr ")"
       | "exp" "(" Expr ")"
       | "log" "(" Expr ")"
       | Variable
       | Number
       ;
```

Otra extensión natural es incorporar relaciones booleanas,

```
boolExpr :: Expr "==" Expr
          | Expr "<" Expr
Expr :: Product ExprRest
ExprRest :: "+" Product ExprRest
         | "-" Product ExprRest
         | "λ"
         ;
Product :: Factor ProductRest
         ;
ProductRest :: "*" Factor ProductRest
            | "/" Factor ProductRest
            | "λ"
            ;
Factor :: "(" Expr ")"
       | Number
       ;
```

En tal caso, nos vemos con el problema de factorizar por la izquierda, esto es, tiene el problema de no saber decidir que regla aplicar en el proceso Top-Down-Parser, ya que solamente aparece con un token lookahead. En tal caso, la pregunta es ¿Cuál de las reglas deberé aplicar?. Para abordar este problema, los analizadores LL(1) determinan que regla aplicar en cada paso en función del token que se encuentra en cada momento en la cabeza de lectura, mientras que los analizadores LL(k), determinan que regla de producción aplicar en cada paso en función de los k primeros tokens que se encuentra en cada momento en la

cabeza de lectura. Para visualizar esta situación consideremos la producción.

```
boolExpr :: Expr “==" Expr
          | Expr “<" Expr
        ;
```

la que se ha transformado en esta otra, la cual ya no tiene esta “ambiguedad” al momento de aplicar la regla.

```
boolExpr :: Expr boolExprRest
boolExprRest :: “==" Expr
              | “<" Expr
             ;
```

El “lookahead” en la generación de parser en ANTLR es una situación no-trivial, ya que tal como se muestra a continuación para la gramática `grammar Left`, la cual presenta una gramática con lookahead de valor k=1, que no deja generar un parser.

```
grammar Left;
options {
k = 1;
}
a : 'x' b
  | 'x' c
  ;
b : 'y' ;
c : 'z' ;
WS: (' '|'\n'|'\r'|'\t')+ { skip(); };
```

La Figura 2 muestra una visualización en ANTLRWorks, la que nos alerta sobre esta especificación.

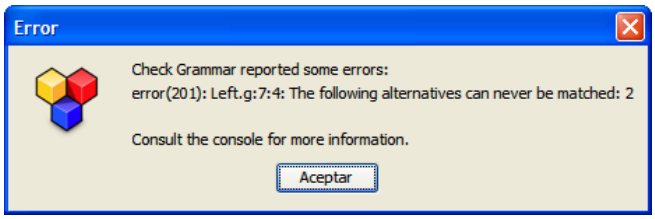

Figura 2. En ANTLRWorks, reporte de error.

A continuación se muestran otros tipos de mensajes de alerta sobre la especificación

```
[14:40:50] warning(200): Left.g:7:4:
Decision can match input such as "'x'" using
multiple alternatives: 1, 2
As a result, alternative(s) 2 were disabled for
that input
[14:40:50] error(201): Left.g:7:4: The following
alternatives can never be matched: 2.
```

Sin embargo, al reemplazar el valor de k=1 por k=2, el problema se supera. Notar que lo ocurrido motiva a que en teoría se traten los conjuntos `First()` y `Follow()`, ya que éste tipo de situaciones no son tan excepcionales, pues bien podría aparecer en una simple declaración de función en C, tal como se muestra a continuación

```
int hola(int a, int b);
int hola(int a, int b){
return 17;
}
```

Ahora, se reescribe la gramática EBNF `expr`, como, `expr_sin` y almacenada en `expr_sin.g`, con el fin de visualizar y analizar su validez con ANTLRWorks.

```
grammar expr_sin;
expr : product (('+'|'-') product)* ;
product : factor (('*'|'/') factor)* ;
factor : '(' expr ')'
       | NUMBER
       ;
NUMBER : ('1'..'9')('0'..'9')*;
WS : (' '|'\t'|'\n'|'\r') { skip(); };
```

Sin embargo, nos encontramos con el mensaje siguiente,

```
warning(138): expr_sin.g:0:0: grammar expr_sin: no
start rule (no rule can obviously be followed by
EOF).
```

La razón obedece a que la especificación ANTLR no sabe distinguir cuál es el símbolo de partida, ya que `expr` aparece tanto al lado izquierdo como derecho de la regla. Es así como incorporando la nueva regla “`start: expr;`”, se soluciona el problema. La Figura 3 muestra la gramática extendida y los diagramas sintácticos asociado a cada regla que ofrece ANTLWorks.

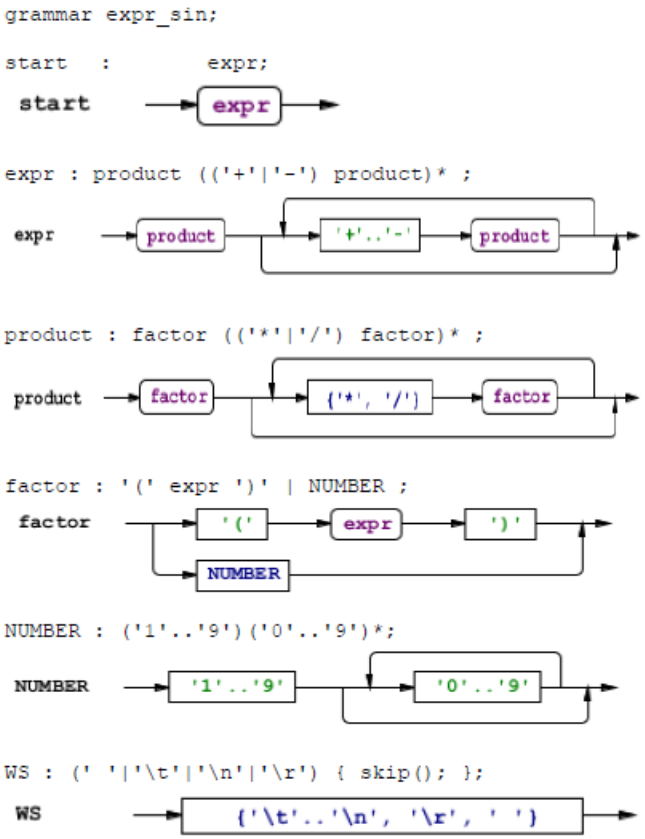

Figura 3. Producciones junto con su diagrama sintáctico.

Una vez definida y testeada la gramática ANTLR genera automáticamente los archivos,

a) `expr_sinParser.java`, que es una extensión de la clase Parser,

b) `expr_sinLexer.java`, que es una extensión de la clase Lexer,

c) `expr_sin.tokens`, este último archivo ordena los token en los enteros para cuando la función del parser desea utilizarla. Finalmente, se necesita una clase testeadora `Test_Expr.java` , tal como se muestra a continuación

```
import org.antlr.runtime.*;
public class Test_Expr {
public static void main(String[] args) throws
Exception {
ANTLRInputStream input = new
ANTLRInputStream(System.in);
Expr_sinLexer lexer = new expr_sinLexer(input);
CommonTokenStream ts = new
CommonTokenStream(lexer);
Expr_sinParser parser = new expr_sinParser(ts);
parser.expr(); //expr, corresponde al símbolo de
partida de la gramática}}
```

Para analizar los datos de salida se incorporan "acciones" o atributos a las gramáticas con especificación ANTLR, es así como se propone la extensión de la gramática `expr_sin.g` con el desarrollo de un "parsers" para el cálculo de expresiones aritméticas, cuyas acciones están embebidas dentro de la gramática que llamaremos `expr_con.g`

```
grammar expr_con;

start   :       expr   {  System.out.println("Valor
de la Expresion: "+$expr.retval); } ;

expr returns [int retval]
 : a=multExpr{$retval = $a.retval;}
  (op=('+'|'-')
   b=multExpr{
if($op.text.equals("+")) $retval += $b.retval;
   else if($op.text.equals("-"))
         $retval -= $b.retval;
             } )* ;

multExpr returns [int retval]
 : a=atom{$retval = $a.retval;}
  (op=('*'|'/')
   b=atom{
if($op.text.equals("*")) $retval *= $b.retval;
   else if($op.text.equals("/"))
         $retval /= $b.retval;
             } )* ;

atom returns [int retval]
 : INT{$retval=Integer.parseInt($INT.text);}
 | '(' expr ')' {$retval = $expr.retval;} ;

 INT : '0'..'9'+ ;
 WS : (' '|'\t'|'\n'|'\r')+ {skip();} ;
```

A continuación, Figura 4, muestra el Parse Tree que se obtiene en ANTLRWorks con la entrada: `2*3+(4-5)` y la salida efectiva con resultado "`Valor de la Expresión. 5`".

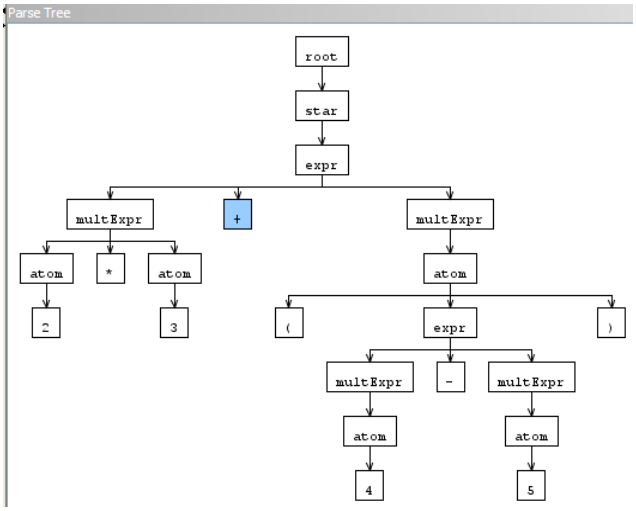

Figura 4. Visualización del Parse Tree (ANTLRWorks).

Otro tipo de "acciones" que pueden incluirse son funciones o métodos nativos de Java, (en este caso `Math.pow();`). En particular, las reglas de elevar a potencia y evaluar una expresión con datos `Double` se muestra a continuación.

```
// fac: factor
fac returns[double vlr] : a=val
{ $fac.vlr = $a.vlr; }
         ( '^' b=val
{ $fac.vlr = Math.pow($fac.vlr,$b.vlr); })*;

// val: valor
val returns[double vlr] : n=NUMERO
{ $val.vlr = Double.parseDouble($n.text); }
        | '(' exp ')'
        ;
```

También, se pueden incluir clases de la librería Swing. Por ejemplo, entrada y salida con uso de `JOptionPane`, tal como se visualiza en Figura 5, a través de la acción embebida en el código de la gramática.

```
multExpr returns [int value]
 :   e=atom {$value = $e.value;}
  ('*' e=atom {$value *= $e.value;}|  '/' e=atom
       { if($e.value!=0)$value /= $e.value;
         else
         JOptionPane.showMessageDialog(null,
"División por cero!! ","Mensaje
Error",JOptionPane.ERROR_MESSAGE); }
```

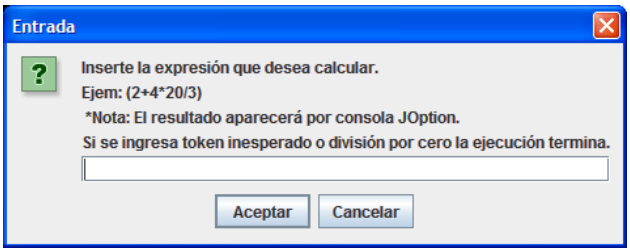

Figura 5. Acción embebida a través de `JOptionPane`.

Por otro lado, en el ámbito de los AST, ANTLR permite construir una gramática de árboles, `tree grammar MiGra`, declarada sobre un archivo `MiGra.g`, la cual procesa un AST y las formas de visitar los nodos. Por ejemplo, dado el siguiente programa `ex1.calc` (palabra de un cierto lenguaje)

```
//ex1.calc
var n: integer;
var x: integer;
n:= 2+4-1;
x:= n+3+7;
print(x);
```

el que se puede generar mediante la siguiente gramática, sin embargo, para ello se deben introducir algunos otros elementos en la gramática de árbol, como por ejemplo, los simbolos "^" y "!".

```
program: decls stats EOF-> ^(PROGRAM decls stats);
decls : (decl SEMICOLON!)* ;
stats : (stat SEMICOLON!)+ ;
decl : VAR^ ID COLON! type ;
stat : assign |  print ;
assign : lvalue BECOMES^ expr ;
lvalue : ID ;
expr : oper (( PLUS^ | MINUS^) oper)* ;
oper : ID | NUM | LPAREN! exp RPAREN! ;
type : INT ;
```

y cuyo árbol AST resultante corresponde al de la Figura 6

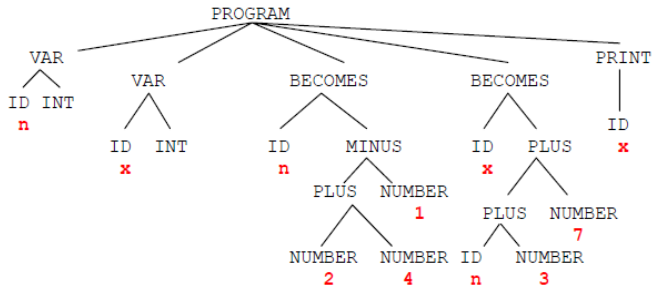

Figura 6. AST tras la lectura del programa `ex1.calc`.

No cabe duda que, dadas las características de ANTLR se hace necesario profundizar en teoría las especificaciones o formato de las gramáticas de contexto libre, algoritmos de eliminación de reglas recursivas por la izquierda, factorización por la izquierda, forma normal de Chomsky, forma normal de Greibach, generadores de análisis sintáctico (parser) mediante descenso recursivo (backtracking, con retroceso), LL(k) (nº de símbolos de preanálisis necesarios) con "lookahead" arbitrario, los conjuntos FIRST y FOLLOW, entre otros. Ahora, desde el punto de vista del árbol sintáctico, el concepto de Autómata a Pila (o PushDown) es fundamental, haciéndose necesario usar etapas del algoritmo de Earley, para parsear la sentencia de acuerdo a la gramática compilada.

V. OBJETIVOS Y EVALUACION

La experiencia muestra que no existe un único modelo de enseñanza, sino que cada académico elabora sus propias estrategias basado en sus "buenas prácticas docentes" para abordar el proceso formativo, y con ello, estar en permanente análisis de los objetivos alcanzados dentro de su práctica docente y los objetivos de la asignatura [27]. En nuestro caso, las buenas prácticas se refieren a Planificar y Ejecutar las actividades, Capturar evidencias y Revisar los procesos continuamente. Esto permite, según los primeros sondeos, realizar los primeros ajustes dentro de la Planificación. La Planificación y Ejecución de las actividades a realizar se sustentan básicamente en documento o material de trabajo generado por el académico, el que es presentado y discutido en las sesiones de Laboratorio, dejando claramente tipificado los objetivos que se esperan lograr para cuando se haya planteado la actividad. Por otra parte, las evidencias son capturadas en el Portafolio, el que contiene la solución del problema propuesto, opiniones y conclusiones por parte de los alumnos frente a las actividades encomendadas, este material forma parte de los insumos que ayudan a mejorar o activar con mayor o menor frecuencia las propuestas de ajuste, para la siguiente actividad. Cabe mencionar que el Portafolio es una forma para recopilar la información que demuestra las habilidades y logros de los estudiantes, capacidades para analizar, sintetizar, producir y crear, es decir, permite identificar los aprendizajes de conceptos, procedimientos y actitudes de los estudiantes[11]. La experiencia bajo esta modalidad permite monitorear el proceso de aprendizaje de manera que pueda introducirse "propuestas de ajuste" durante el proceso. No obstante, la metodología que subyace en las actividades de Laboratorio se sustenta en una variante del aprendizaje basado en problemas, ya que la diferencia estriba en que aquí son impartidos los contenidos básicos que orientan hacia la solución de la actividad, en contra posición a la que el alumno descubre y trabaja con el contenido que él determina necesario para resolver el problema [28]. El alumno trabaja en forma individual (sobretodo en las primeras actividades, para luego hacerlo en forma grupal de 2 o 3 alumnos) para resolver de manera incremental las dificultades que se le van incorporando a los problemas durante el semestre. Para entonces, ya se tienen definido los objetivos fundamentales, por ejemplo:

- Generar y analizar el comportamiento de un parser,
- Desarrollar sus propias visualizaciones,
- Observen, manipulen e intervengan el AST,
- Codificar la gramática utilizando el editor de ANTLRworks,
- Uso del intérprete para visualizaciones estáticas,
- Uso del depurador para ver el proceso de animación,
- Codificar la gramática utilizando un editor de propósito general,

- Cambiar la precedencia de los operadores binarios de una gramática,
- Visualización del árbol sintáctico y su proceso de construcción,
- Incorporar los conceptos de la POO y la arquitectura MVC(Model-View-Controller) en algunas aplicaciones.

En términos administrativos, las actividades de Laboratorio tienen una duración de 1 o 2 semanas, dependiendo de la dificultad prevista por el profesor. Al cabo de ese tiempo, el alumno expone y defiende su propuesta, para posteriormente entregar un informe preliminar en formato Portafolio y modalidad "showcase", respecto a la actividad encomendada. En éste informe preliminar, el alumno debe incorporar dentro del mismo las observaciones que surgieron en la defensa, ya sea por los compañeros o por el profesor, pudiendo incluso mejorar la versión original. Todos estos insumos son básicos y fundamentan la calificación respecto a la actividad puntual, cómo también la evaluación del Laboratorio, al término del semestre.

VI. ACTIVIDADES

Este segmento muestra las Actividades propuestas en los Laboratorios, junto con las opiniones y conclusiones planteadas por algunos alumnos en el Portafolio.

Actividad nº1

a) Instalación, manejo y puesta a punto de cada una de las herramientas, esto es, Lex/Yacc, Flex/Bison, ANTLR.
b) Construya un analizador lexicográfico básico en cada una de las herramientas antes señaladas.
c) Construya un analizador sintáctico básico en cada una de las herramientas antes señaladas.

Conclusiones Actividad nº1

La experiencia consistió en la instalación de ANTLR y poner a punto la herramienta con la creación de una gramática básica, que reconoce una expresión matemática con operadores de suma, resta, multiplicación y división. El uso de ANTLR en la solución de problemas es tan variada que puede incluir otros ámbitos, sobre todo en lo que trata sobre la intercomunicación de sistemas ya que ANTLR brinda la posibilidad de crear un lenguaje o código intermedio entre sistemas. También se observa que la herramienta de ANTLRWorks es de bastante ayuda ya que facilita la tarea de crear gramáticas fácilmente, un entorno bastante intuitivo y con una gran facilidad de instalación, si la comparamos con otras herramientas, tales como, Flex o Bison en los cuales la instalación puede ser muy complicada, además que no se cuenta con un IDE de apoyo, como sí lo es ANTLRWorks.

Actividad nº2

a) Instalación plug-in ANTLR, manejo y puesta a punto de la herramienta ANTLRWorks, en NetBeans y Eclipse.
b) Construya un analizador lexicográfico básico para las expresiones aritméticas en ANTLRWorks y visualice sus propiedades a partir de las herramientas que pone a disposición ANTLRWorks.
c) Hacer las pruebas necesarias para su correcta ejecución desde ANTLR. Por ejemplo, que tras el comando

```
>java Test
4+5*5
(4+5)*5
x=5
y=10
x*y
^Z
```

La salida sea:

```
29
45
50
```

d) Use los comandos apropiados para que los datos de entrada (resp. salida) que alimenten a la gramática se tomen de un archivo de texto plano.
e) Intervenir la precedencia de los operadores. Por ejemplo, en la expresión 1 + 5 * 3, la respuesta es 16 y no 18, ya que el operador de multiplicación ("*") tiene una mayor precedencia que el operador de adición ("+"). Los paréntesis pueden ser usados para marcar la precedencia, si resulta necesario. Por ejemplo: (1 + 5) * 3 evalúa a 18. Si la precedencia de los operadores es la misma, se utiliza una asociación de izquierda a derecha. Basado en esta situación, cambie la precedencia a la gramática de las expresiones aritméticas.

Conclusiones Actividad nº 2

En esta actividad se ingresaron algunas acciones básicas en la gramática, entre ellas, intervenir la precedencia, con esto queda claro de las facilidades de ANTLR al poder incorporar nuevas reglas que permitan realizar las asignaciones básicas, operaciones matemáticas más complejas, operadores relacionales, booleanos, etc. pudiendo ocasionar que se transforme una gramática en una secuencia no despreciable de reglas.

Actividad nº3

Dada una gramática `ejemplo.g` especificada según ANTLR. Se pide que la extienda a una gramática `ejemplo1.g` que incluya las operaciones de "multiplicación"(*), "división"(/) y "elevar a potencia" (^, por ahora es suficiente incluirla a través de `Math.pow();`). Al igual que en la Actividad nº2 cambie la precedencia a la gramática `ejemplo1.g`. Una vez que obtenga la gramática `ejemplo1.g`, se pide que le incluya algunas "acciones". Por ejemplo, "acciones" que las conecten con la clase `JOptionPane` de Swing para ingresar y mostrar los resultados.

Conclusiones Actividad nº 3

En este laboratorio se profundizó en la sección de modificar las clases generadas por el parser, (archivo `xxParser.java`) y no la clase principal `.java` o la gramática `xx.g`, con lo

cual se pudo comprobar que añadir funcionalidades mas avanzadas a una gramática se relaciona con el uso de clases nativas Java o clases propias de Java. En este caso se hizo una modificación en la clase del parser para poder invocar métodos de la librería `Collections` y lograr ordenar los elementos de un `ArrayList` que se uso en la gramática. Se pudo demostrar que ANTLR permite usar diversas estructuras de datos, y capacidad para almacenar valores que usa la gramática y así usarlas posteriormente. En esta actividad, se almacenaron valores numéricos en un `ArrayList` (ya que esta estructura permite guardar números enteros, pero bien podría haberse usado `TreeMap` o alguna otra). En otras actividades se almacenaron los elementos en un `HashMap`, que permite guardar en una misma estructura tanto valores numéricos como valores alfanuméricos.

Actividad nº4

Existen tres formas clásicas de leer o expresar una expresión matemática, ellas son: infija, prefija y postfija. Los prefijos, (Pre - Pos - In) se refieren a la posición relativa del operador con respecto a los dos operandos. **Ejemplo:** Una de las más familiares notaciones son la infija, en donde si desea sumar 5 más 4; en notación infija quedaría: 5+4, en notación prefija quedaría: + 5 4, y finalmente, en notación postfija: 5 4 +. Se pide incluir una lectura (Pre - Pos - In) en alguna de las gramáticas que obtuvo de las actividades anteriores.

Conclusiones Actividad nº 4

El tema da para admitir que una gramática puede soportar diferentes formas de reconocer una misma cadena con el mismo resultado, por lo cual reconocer `+ 5 6, 5 6 +, 5 + 6`, resulta ser muy simple. Esto permite crear gramáticas con tolerancia a los errores, por lo cual una gramática bien hecha podría reconocer una operación matemática bien escrita y a la vez una que este mal escrita pero que a la vez sea entendible.

Actividad nº5

En una empresa de seguros se desea procesar un archivo plano con valores del tipo o formato,

```
4564,Tiempo:Jugar y Pagar,1:00,1,"41,38","258,62".
```

Esto es solo el comienzo, ya que posteriormente se desea llevarlo a un formato de Tablas en HTML. Como se visualiza en el patrón dado, la dificultad surge a través de la separación por la coma, debido a que existen nº decimales que también están separado por coma. Se pide que lo solucione con los conocimientos y praxis que ya ha tenido en ANTLR. Luego de solucionado el problema anterior, se pide que use las librerías de Swing, para llevar los datos a `JTable`, en donde las columnas queden bien delimitadas, como una solución intermedia al formato HTML.

Conclusiones Actividad nº 5

Esta actividad permitió manejar todas las herramientas y conocimientos adquiridos hasta ahora. Las plataformas hasta ahora utilizadas para la realización del laboratorio son dos. La primera es utilizando solamente ANTLR / ANTLRWork, generando los archivos desde esta herramienta y compilando los archivos generados en Java desde la consola, en la clase de testeo de esta actividad agregamos elementos nativos de Java como son `JTable, JFileChoser, JOptionPane`, ya sea para ingresar el texto a parsear o cargarlo desde un archivo de texto. La segunda es a través del framework NetBeans que requiere de la integración con ANTLR, la que ayuda en la generación automática de los archivos y la ejecución de la clase de testeo. Se volvió a tratar de incorporar Eclipse, lo que no fue posible en los tiempos previstos. Tal vez sea que nos hemos acostumbrado con NetBeans, plataforma por defecto de las asignaturas POO y DAA. Respecto al proceso de incluir acciones, se abordaron de dos formas:

a) Se incluye código Java dentro de la misma gramática, esto es, usando una clase externa de Java como lo es la creación de un documento con formato `xls`, esta clase se importa como una librería en la gramática. De esta forma se puede usar de forma libre sus métodos y variables, para generar un nuevo documento Excel con los tokens reconocidos por la gramática.

b) Otra alternativa es no incluir código Java dentro de la misma gramática, sino intervenir el archivo `xxParser`, que es una extensión de la clase Parser generada por ANTLRWorks, agregando en él una clase que sea capaz de retornar lo que necesitamos desde el parser a la clase de testeo, para ello cargamos en una `List` los datos que necesitamos del parseo del documento, para retornar a la clase testeadora y agregar directamente al `JTable` desde la lista. Los conocimientos de `Generic`, `Collections` y `Swing`, junto con otras librerías, han sido fundamentales para seguir incorporándole acciones a las gramáticas. Parece razonable pensar en la construccción de una transformación, en el sentido de pasar de un lenguaje de dominio visual a otro textual con el uso de la librería `Graphics` de Java. En éste contexto, parece ser razonable concluir que no es un problema trivial el de incorporar acciones a la gramática.

Conclusiones generales:

a) A través del uso de ANTLR/ANTLRWorks podemos reconocer y construir reconocedores sintácticos, por ejemplo, crear asistentes de voz (tal como la tecnología SIRI de Apple) con posibilidades infinitas y que no están mayormente difundidos en el mundo de las Tecnologías de la Información y Comunicación (TIC). Por lo cual nos anima el crear un posible asistente de voz para notebooks o celulares Android (cuya lenguaje de programación es efectivamente Java) en una posible memoria de título. En lo particular, veo una aplicación en personas con alguna discapacidad física o motora, las que no pueden manejar de forma completa un computador, y cuyas ayudas solo vienen por pobres asistentes de voz.

b) Hacer uso de expresiones regulares para la validación de datos. Por ejemplo, validar sentencias del lenguaje SQL (u

otro, subconjunto de él) y trasladarla a sentencias XML, HTML, LATEX, etc.
c) Aplicar los algoritmos de grafos vistos en DAA, tales como Kruskal, Johnson y otros, cuya entrada/salida de datos, usualmente listas o matrices de adyacencia, puedan ser transformadas previamente a formato `.dot` de GraphViz.

Cabe mencionar que en esta sección, las conclusiones se han transcrito de la forma más fiel a la opinión vertida por los propios alumnos, y que van desde las dificultades hasta algunas ideas de transformación de gramáticas y aplicaciones de algoritmos.

VII Conclusiones

Las actividades fueron realizadas en los tiempos previstos y los objetivos logrados por un 90% de los alumnos (de un total de 11 alumnos) de la asignatura TALF, al mismo tiempo se constata que han alcanzado un manejo de muy buen nivel en lo que se refiere a la programación orientada a objetos en Java. A la luz de los resultados, se constata que las herramientas propuestas ANTLR/ANTLRWorks ayudan a entender de mejor manera los conceptos teóricos impartidos, entre ellos, los analizadores lexicográficos y sintácticos, así como el trabajo o funcionalidad de los autómata a pila como reconocedores de las gramáticas de contexto libre LL(k). Los resultados obtenidos forman parte de un repositorio de experiencias y evidencias docentes al término del semestre, ya que esta actividad está en pleno desarrollo. Se continúa con el programa del curso con énfasis en LL(k), es así como algunos desafíos pendientes es lograr la construcción de un traductor simple, usando atributos y acciones semánticas (basada en clases de Java) embebidas dentro de la gramática ANTLR. No cabe duda que a la luz de las conclusiones generales por parte de los alumnos, ya se tiene un buen insumo de ideas para concluir el semestre, y por que no, pueda surgir incluso la propuesta de una memoria de título. Sin embargo, en lo particular, se pretende integrar librerías externas en una gramática `Exp.g` "pura", la cual pueda ser modificada en el ámbito AST, llamada `Tree.g` y dada su especificación permita que ante una entrada del tipo `x*y+(x+y)*z`, (la que se encuentra almacenada en el archivo `input.txt`) genere un archivo del tipo `.dot` de la librería GraphViz, en conjunto con otras clases Java.

En base a la experiencia de estos 2 años, somos de la opinión que ANTLR/ANTLRWorks es una herramienta didáctica y factible de incorporar en un curso de pregrado, pero con la precaución de reforzar y/o enfatizar en temas más específicos, como los expuestos en sección IV. Por otra parte, si bien se instaló y se mostró en Actividad nº1 algún ejemplo en JavaCC, Lex/Yacc (Flex/Bison), pero los énfasis iban por instalar, usar y aplicar desde un comienzo las herramientas ANTLR/ANTLRWorks, de ahí que cualquier manera de comparar estas con otros entornos clásicos escapaba de los objetivos planteados. El motivó fue que, si bien, las herramientas clásicas son plausibles, pero la múltiple variedad de código en lenguajes de propósito general que ofrece ANTLR, tales como (Java, C, C++, Ruby y otros) es una de las ventajas que bien deben ser consideradas, sobretodo para construcción de gramáticas o lenguajes DSL en entornos o escenarios en donde los lenguajes de propósito general bien no pueden llegar, o al menos no con la prontitud con que lo hace ANTLR. Además, la existencia de un editor de gramáticas y depurador (debugger) como ANTLRWorks facilita aún más la tarea. Sin embargo, lo más importante de trabajar con estas herramientas es la activa y entusiasta comunidad, liderada por Terence Parr, Jean Bovet y muchos otros….

VIII Agradecimientos

No puedo dejar de agradecer el apoyo del grupo de curso de la asignatura TALF_2012 (1º Semestre) por haber tenido tan buena disposición, colaboración y ánimo de participar en esta propuesta.

IX Referencias

[1] Samuel's Project Blog. URL: http://saml.rilspace.org/ebnf-parser-for-parts-of-the-galaxy-toolconfigs-with-antlr. Fecha de consulta: 12 Julio 2012.

[2] I. Martínez, J. Fernández, M. Galarraga, L. Serrano, P. de Toledo and J. García, "Implementation of an End-to-End Standards-based Patient Monitoring Solution," IET Communications - Special Issue on Telemedicine and e-Health Communication Systems, vol. 2, pp. 181-191, 2008.

[3] L. Hawizy, D. M. Jessop, N. Adams, P. Murray-Rust. ChemicalTagger: A tool for semantic text-mining in chemistry. Journal of Cheminformatics 2011, 3:17. 2011.

[4] M. Liu, A. Grigoriev. Fast parsers for Entrez Gene. Data and text mining,Vol. 21 no. 14, pp. 3189–3190, doi:10.1093/bioinformatics/bti488. 2005.

[5] ANTLRv3. http://www.antlr.org/. Fecha de consulta: 12 Julio 2012.

[6] A.Sarasa-Cabezuelo, J.L.Sierra-Rodríguez, A. Fernández-Valmayor, Procesamiento de Documentos XML Dirigido por Lenguajes en Entornos de E-Learning, IEEE-RITA, Vol. 4, Núm.3, pp.175-183, 2009.

[7] J. Bovet, T. Parr. ANTLRWorks: an ANTLR grammar development environment. Software: Practice and Experience, 38(12): 1305-1332, 2008.

[8] M. Watfa, E-Healthcare Systems and Wireless Communications,IGI Global, October 31, 2011, DOI: 10.4018/978-1-61350-123-8.

[9] J. Escayola. Contribución a estándares y tecnologías de comunicación en dispositivos médicos para e-salud: Integración en aplicaciones de Telemonitorización y gestión de la información. URL: http://zaguan.unizar.es/record/7489/files/TESIS-2012-073.pdf. Fecha de consulta: 12 Julio 2012.

[10] L. Gajardo, L. Mateu. Análisis Semántico de programas escritos en Java,.Theoria, Vol. 13: 39-49. 2004.

[11] P.Paulson, F.Paulson. Portfolios: Stories of Knowing. URL: http://www.eric.ed.gov/PDFS/ED377209.pdf. Fecha de consulta: 12 Julio 2012.

[12] T. Parr: The Definitive ANTLR Reference: Building Domain-Specific Languages. Raleigh: The Pragmatic Bookshelf, 2007.

[13] F. Almeida. Evaluación educativa y de usabilidad de VAST, ANTLR y ANTLRworks. SITIAE'10. Seminario de Investigación en Tecnologías de la Información Aplicadas a la Educación. http://www.lite.etsii.urjc.es/sitiae/2010/ Fecha de consulta: 12 Julio 2012.

[14] J.Urquiza-Fuentes, F.Almeida-Martínez, A. Pérez-Carrasco. Reorganización de las prácticas de compiladores para mejorar el aprendizaje de los estudiantes. JENUI-2010. Jornadas de Enseñanza Universitaria de la Informática.
http://www.aenui.net/ActasJENUI/2010/Jenui2010_11.pdf . Fecha de consulta: 12 Julio 2012.

[15] M. Aguirre, E.Jeltsch, G.Rosales. Una visualización de un Sistema de Inferencia a través de TreeBaG, Ingeniare, Revista Chilena de Ingeniería, vol. 15 nº 1, pp. 92-100. 2007.

[16] M. Aguirre, E. Jeltsch, G. Rosales. Generación de formas pictóricas sobre un sistema de Inferencia a través de TreeBag. X Encuentro Chileno de Computación. Copiapó, Chile. 2002.

[17] M. Aguirre E. Jeltsch, , G. Rosales. Aplicaciones en TreeBag. IV Congreso de Educación Superior en Computación. Copiapó, Chile. 2002.

[18] E. Jeltsch, S. López, W. Villacorta, Herramientas para la implementación de un sistema de Telemedicina, desde el punto de vista de la Ingeniería de Software. INFONOR-Chile 2010, Antofagasta, Chile. 2010.

[19] E. Jeltsch, S. López, W. Villacorta. El proyecto SAT: Desarrollo de un Sistema de Telemonitorización de pacientes. INFONOR-Chile 2010, Antofagasta, Chile.2010.

[20] ACM. Computing Curricula 2005.
http://www.acm.org/education/education/curric_vols/CC2005-March06Final.pdf. Fecha de consulta: 12 Julio 2012.

[21] StringTemplate. http://www.stringtemplate.org/. Fecha de consulta: 12 Julio 2012.

[22] M.Fowler, Introducing Domain-Specific Languages. DSL Developer's Conference; http://msdn.microsoft.com/en-us/data/dd727707.aspx. Fecha de consulta: 12 Julio 2012.

[23] D. Ghosh, ACMQueue. DSL for the uninitiated: Domain-specific languages bridge the semantic gap in programming,
http://queue.acm.org/detail.cfm?id=1989750. Fecha de consulta: 12 Julio 2012.

[24] HL7. Health Level 7 - IEEE interoperability JWG. http://www.ieee1073.org/ jwg/hl7ieeeinterop.html. Fecha de consulta: 12 Julio 2012.

[25] R. Soto. http://www.inf.ucv.cl/~rsoto/ICI445.php. Fecha de consulta: 12 Julio 2012.

[26] F. Almeida-Martínez, J. Urquiza-Fuentes. Building LL (1) grammars and parsers using VAST and ANTLR.
http://eciencia.urjc.es/bitstream/10115/5521/1/DLSI1-URJC_2011-10.pdf. Fecha de consulta: 12 Julio 2012.

[27] Taxonomy of Educational Objectives: The Classification of Educational Goals; pp. 201-207; B. S. Bloom (Ed.) David McKay Company, Inc. 1956.

[28] L. Spence. The Usual Doesn't Work: Why We Need Problem-Based Learning, Libraries and the Academy, Volume 4, Number 4, pp. 485-49. 2004.