SISTEMA DE TRADUCCIÓN DE COMANDOS DE CISCO A AVAYA*

Por: Darwin Ramiro Mercado Polo**

Fecha de recibido: 1 de julio de 2010 • Fecha de aceptación: 30 de septiembre de 2010

RESUMEN:
Es especialmente relevante para la formación de los ingenieros de sistemas y profesionales de carreras afines, adquirir las competencias relacionadas con la configuración de dispositivos de comunicaciones presentes en redes informáticas, por tanto es muy importante que el estudiante aprenda a manejar una gran variedad de comandos aplicables en tales dispositivos. Los comandos varían de un dispositivo a otro, dependiendo de la empresa manufacturera, algunas de ellas son: CISCO System, AVAYA y Nortel Networks. Las arquitecturas desarrolladas por estas empresas son configurables tanto en el ámbito de la LAN como el de la WAN. En este artículo se muestra como resultado de una investigación, un software que posibilita la traducción de comandos de CISCO a AVAYA, teniendo en cuenta que para esta traducción se utilizaron diferentes técnicas y herramientas de los lenguajes formales y gramáticas, aplicando las etapas iniciales de un traductor como: las etapas de análisis léxico y sintáctico.

PALABRAS CLAVE:
Lenguajes formales, Plataforma AVAYA, Plataforma CISCO, Léxico, Sintáctico, Traducción, Redes.

Revista Inge-CUC / Vol. 6 - No. 6 / Octubre 2010 / Barranquilla - Colombia / ISSN 0122-6517

CUC
CORPORACION UNIVERSITARIA DE LA COSTA

* Artículo resultado de la investigación Sistema de traducción de comandos CISCO-AVAYA. Grupo de investigación: Ingeniería de software y redes. Línea de Investigación: Ingeniería de software. Investigador principal: Darwin Mercado Polo.

** Ingeniero de sistemas. Especialista en informática y telemática, especialista en estudios pedagógicos Ph.D (C) Ingeniería de Software. Docente Tiempo Completo del programa de Ingeniería de Sistemas. Calle 19 N°17-27 Sabanalarga. dmercado@cuc.edu.co

CUC

TRANSLATION SYSTEM COMMAND CISCO - AVAYA

By: Darwin Ramiro Mercado Polo

ABSTRACT:

It is particularly relevant to the training of systems engineers and professionals in related careers, acquire skills related to the configuration of communication devices found in computer networks, so it is very important that students learn to handle a wide variety of commands applicable on such devices. The commands vary from one device to another, depending on the manufacturing company, some of them are: CISCO System, Avaya and Nortel Networks. The architectures developed by these companies are configurable at both the LAN and the WAN. This article shows the result of an investigation, a software that enables the translation of commands to AVAYA CISCO, bearing in mind that this translation is used different techniques and tools for formal languages and grammars, applying the initial stages of a translator such as: the stages of lexical and syntactic analysis.

KEY WORDS:

Formal languages, AVAYA Platform, CISCO Platform, Lexicon, Syntax, Translation, Networks.

CUC
CORPORACION UNIVERSITARIA DE LA COSTA

EDUCOSTA
EDITORIAL UNIVERSITARIA DE LA COSTA

1. INTRODUCCIÓN

Este artículo es de gran importancia porque facilitará al docente el proceso enseñanza-aprendizaje y mejoraría la aprehensión del estudiante. Mediante este software se podrá tener como referencia o como base una plataforma la cual debemos conocer muy bien y luego podemos pasar a los comandos de la otra plataforma en una forma sencilla y rápida, es decir, solo es necesario conocer bien los comandos de la plataforma CISCO.

Cuando la persona escriba un comando en la plataforma CISCO inicialmente el software realizará un escaneo de todo el comando, es decir, un análisis léxico y sintáctico. En primera instancia reconocerá cada uno de los componentes que lo conforman como palabras reservadas, variables, constantes o símbolos y después identificará la conformación lógica de toda la instrucción, o sea reconocerá si la instrucción tiene un significado colectivo coherente con la sintaxis de la plataforma a traducir.

Una vez realizado el análisis léxico-sintáctico de la instrucción el software informará si se generó un error mostrando un mensaje que indique exactamente cuál fue el tipo de error y en caso contrario el usuario puede hacer la traducción de comandos y mostrará el equivalente de esa instrucción en la otra plataforma.

2. TRADUCTOR

Un traductor es un programa que toma como entrada un texto escrito en un lenguaje, llamado fuente y da como salida otro texto en un lenguaje, denominado objeto[1].

En el caso de que el lenguaje fuente sea un lenguaje de programación de alto nivel y el objeto sea un lenguaje de bajo nivel (ensamblador o código de máquina), a dicho traductor se le denomina compilador. Un ensamblador es un compilador cuyo lenguaje fuente es el lenguaje ensamblador. Un intérprete no genera un programa equivalente, sino que toma una sentencia del programa fuente en un lenguaje de alto nivel y la traduce al código equivalente y al mismo tiempo lo ejecuta.

Una tarea frecuente en las aplicaciones de las computadoras es el desarrollo de programas de interfaces e intérpretes de comandos, que son más pequeños que los compiladores pero utilizan las mismas técnicas. En los traductores en forma general se deben seguir una serie de etapas que difieren muy poco con respecto a las etapas de un compilador, solo en la etapa de síntesis puede haber variaciones porque la traducción final no siempre es hacia un bajo nivel.

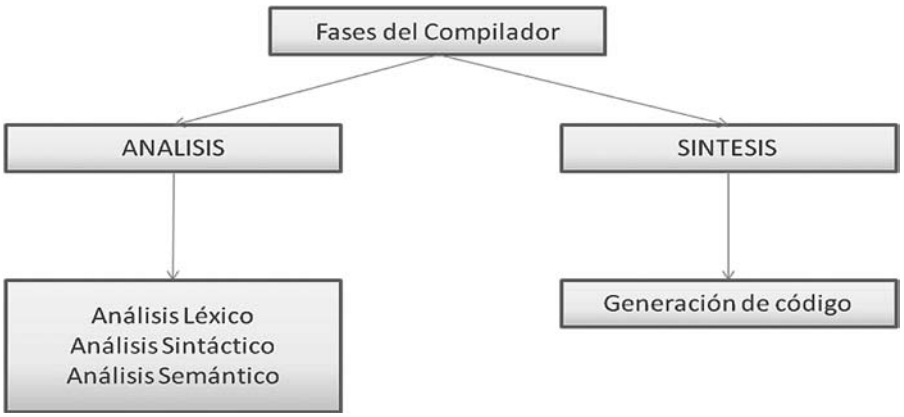
2.1. Fases y etapas del compilador

En general todo compilador debe tener una etapa de análisis y una de síntesis. La primera realiza toda la evolución y verificación del código fuente y la síntesis es donde se realiza la traducción final (Ver figura 1).

En la fase de **análisis léxico** se revisa o se examina el programa fuente como una cadena de izquierda a derecha y se generan los componentes léxicos o token a partir de una secuencia de caracteres que tenga un significado válido. Luego en la fase de análisis sintáctico recibe todos los componentes léxicos reconocidos en la etapa anterior y los agrupa en forma jerárquica a través de la gramática independiente del contexto (define si está bien escrito). Seguidamente en la etapa de análisis semántico el compilador intenta detectar construcciones que tengan la estructura sintáctica correcta, pero que no tengan significado para la operación implicada y por último en la generación de código se toma el programa y se

1. AHO, A. V.; SETHI, R. y ULLMAN, J. D. (1990). Compiladores: Principios, técnicas y herramientas. Wilmington, Delaware: Addison-Wesley Iberoamericana S.A., p. 1.

Figura 1. Fases del Compilador



crea un código intermedio que generalmente lo traen todos los compiladores. La fase de análisis léxico se constituye en la fase de estudio necesaria para la construcción del analizador lexicográfico en cuestión.

3. ANÁLISIS LÉXICO

El analizador léxico es la primera etapa de un traductor. Para este análisis es importante tener en cuenta algunos conceptos como: componentes léxicos o tokens, patrón o regla, lexema, cadena, lenguajes, alfabetos y un concepto muy importante que es el de expresiones regulares el cual permite simplificar la especificación de un lenguaje regular y sirve para especificar un patrón [3] y por último los autómatas finitos los cuales son grafos dirigidos que representan las acciones que tienen lugar al obtener el siguiente componente léxico, además es la estructura que permite representar gráficamente a las expresiones regulares.

Los conceptos anteriormente citados como las expresiones regulares y autómatas finitos son la parte fundamental para el reconocimiento de tokens o componentes léxicos. El proceso de reconocimiento desde el punto de vista teórico requiere de conceptos muy abstractos que surgen de los lenguajes formales, sin embargo existen generadores como Javacc y jflap que facilitan ese reconocimiento o permiten reconocer cadenas en una forma práctica sin necesidad de realizar un seguimiento analítico que implica mucho tiempo de análisis y una excelente apropiación del tema.

3.1. Lenguajes regulares

Si {a} y {$\in$} son lenguajes básicos. Un lenguaje regular en un $\sum$ es uno que puede obtener de esos lenguajes básicos con las operaciones de unión, concatenación y cerradura de Kleene[2].

Definición formal:

1. {$\in$} es un lenguaje regular
2. Si ***a*** pertenece al alfabeto ($\sum$) entonces {***a***} es un lenguaje regular.
3. Si L_1 y L_2 son lenguajes entonces $L_1 L_2$, $L_1 \cup L_2$, L_1^*, L_2^* son lenguajes regulares.

3.2. Expresiones regulares[3]

Simplifica la especificación de un lenguaje regular y sirve para especificar un patrón.

Los operadores que se utilizan son los siguientes: Concatenación, unión, cerradura de

2. MARTÍN, J. (2004). *Lenguajes Formales y teoría de la Computación*. Editorial McGraw-Hill. p. 85.
3. DE CASTRO, R. *Teoría de la Computación: Lenguajes, Autómatas y Gramáticas*. p. 19.

Kleene, cerradura positiva y ceradura de cero o un caso.

Definición:

1. ∈ es una expresión regular.
2. Si ***a*** pertenece a ∑, entonces ***a*** es una expresión regular.
3. Si r y s son expresiones regulares entonces su lenguaje regular es respectivamente L(r) y L(s).
   a. r es una expresión regular y se representa L (r) = {r}
   b. (r | s) es una expresión regular y se representa, L(r) U (s) = {r} U {s} = {r, s}.
   c. r . s es una expresión regular y se representa L(r) L(s) = {r}{s} = {rs}.
   d. r* es una expresión regular y se representa

L*(r) = {r}* = {r, rr, rrr, rrrr…}.

Ejemplo

∑ = {a, b} → Alfabeto
a. b* → Expresión regular

El lenguaje correspondiente a la expresión regular

a. L(a).L*(b) cada uno representa su conjunto, entonces,
   L(a).L*(b)={a}.{b}*, luego se desarrolla la cerradura de Kleene de
   {b}*={ ∈,b,bb,bbb,....} concatenado da como resultado
   {a}.{b}*= {a, ab, abb, abbb, abbbb,.....}.

Este lenguaje regular corresponde a la expresión regular a. b*.

4. ANÁLISIS SINTÁCTICO

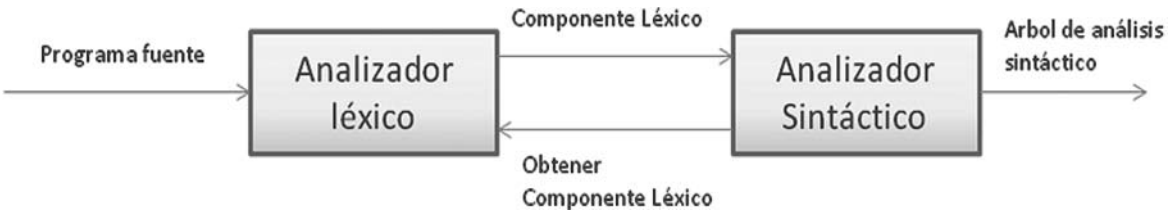
El analizador sintáctico tiene como función obtener una cadena de componentes léxicos del analizador léxico como se muestra en la figura 2 y además comprueba si la cadena puede ser generada por la gramática del lenguaje fuente. El analizador debe informar de cualquier error sintáctico que se presente de manera inteligible[4].

Como muchos de los lenguajes de programación tienen una estructura inherentemente recursiva entonces se pueden definir mediante Gramática Independiente del Contexto (GIC). Esta gramática tiene una gran facilidad para resolver lenguajes regulares y no regulares.

4.1. Gramática independiente del contexto o Gramática BNF (Forma Backus-Nour)[5]

1. **Terminal:** Símbolo básico a partir del cual se pueden formar las cadenas, ejemplo: una letra, palabra reservada, *, etc.
2. **No terminal:** Es una variable sintáctica que puede ser sustituida por una terminal, una expresión u otra variable sintáctica.

Figura 2. Posición del analizador sintáctico en el traductor



4. AHO, A. V.; SETHI, R. y ULLMAN, J. D. Compiladores: Principios, técnicas y herramientas. Wilmington, Delaware: Addison-Wesley Iberoamericana S.A., 1990. p. 164.
5. KENNETH, L. C. *Construcción de Compiladores. Principios y práctica*. México: Thomson editores, 2004. p. 34.

3. **Símbolo inicial:** Es un símbolo no terminal a partir del cual se origina la gramática.
4. **Producción:** Expresión que asocia terminales y no terminales.

4.2. Analizadores sintácticos[6]

Existen varios tipos generales de analizadores sintácticos para gramáticas como los métodos universales Cocke-younger-Kasami y el de Earley, estos métodos son muy ineficientes para usarlos en la producción de compiladores. Los más empleados se clasifican en descendentes y ascendentes.

La función principal de estos analizadores sintácticos consiste en examinar la secuencia de tokens entregados por el analizador léxico y verificar si es sintácticamente correcta mediante la obtención del árbol de derivación asociado a dicha secuencia.

4.2.1. Analizador sintáctico descendente

- Son llamados predictivos y orientados hacia un fin, debido a la forma en que trabajan y construyen el árbol sintáctico.
- Construyen al árbol sintáctico de la sentencia a reconocer de una forma descendente comenzando por el símbolo inicial o raíz, hasta llegar a los símbolos terminales que forman la sentencia.
- Trabaja con la gramática LL(1).

4.2.2. Analizador sintáctico ascendente

Trabaja la gramática LR.

Donde:

- R: Derivación derecha (derecha invertida).
- L: Evaluación de izquierda a derecha.

- También trabajan con las siguientes gramáticas, que son mejoras de la anterior: **gramática SLR y gramática LALR**.
  - SLR: LR sencillo.
  - LALR: LR con análisis anticipado.

En el desarrollo de este traductor se utiliza una gramática LL, ya que es la gramática que utiliza Javacc en el proceso de traducción.

4.3. Gramática LL[7]

La gramática LL permite construir de manera automática analizadores sintácticos predictivos. La ventaja se debe al hecho de poder construir manualmente analizadores sintácticos eficientes con mayor facilidad.

Esta gramática utiliza un análisis sintáctico predictivo en el que el símbolo de pre-análisis determina sin ambigüedad el procedimiento seleccionado para cada no terminal.

La gramática necesita para su proceso predictivo una gramática muy elaborada, es decir no debe tener ambigüedad, ni recursividad por izquierda, ni factorizable por izquierda.

5. ANTECEDENTES DE TRADUCTORES

En la actualidad existen muchos manuales para aprender y aplicar los comandos de la plataforma CISCO y AVAYA, sin embargo no existen programas que enseñen la traducción de una plataforma a otra. En algunas consultas realizadas en medios digitales y bibliográficos no se pudieron encontrar antecedentes de este tipo de traductor. Se encontró una traducción de comando de CISCO denominado **NAT** (Network Address Traslation) y **PAT** (Port Address Traslation)[8].

NAT (Network Address Traslation) él permite acceder a Internet traduciendo las direcciones privadas en direcciones IP registradas. Incrementa la seguridad y la privacidad de la red local al traducir el direccionamiento interno a uno externo.

6. KENNETH, L. C. *Construcción de Compiladores. Principios y práctica*. México: Thomson editores, 2004. p. 34.
7. AHO, A. V.; SETHI, R. y ULLMAN, J. D. Compiladores: Principios, técnicas y herramientas. Wilmington, Delaware: Addison-Wesley Iberoamericana S.A., 1990. pp. 189-190.
8. ARIGANELLO, E. Blog CCNA *Aprende Redes en www.aprenderedes.com*

PAT (Port Address Traslation): Es una forma de NAT dinámica que asigna varias direcciones IP internas a una sola externa. PAT utiliza números de puertos de origen único en la dirección global interna para distinguir entre las diferentes traducciones.

5.1. Plataforma CISCO y AVAYA

Cisco Systems es una empresa multinacional ubicada en San José (California, Estados Unidos), principalmente dedicada a la fabricación, venta, mantenimiento y consultoría de equipos de telecomunicaciones tales como:

- Dispositivos de conexión para redes informáticas: routers (enrutadores, encaminadores o ruteadores), switches (conmutadores) y hubs (concentradores);
- Dispositivos de seguridad como Cortafuegos y Concentradores para VPN;
- Productos de telefonía IP como teléfonos y el CallManager (una PBX IP);
- Software de gestión de red como CiscoWorks, y
- Equipos para *redes de área de almacenamiento*.

Actualmente, Cisco Systems es líder mundial en soluciones de red e infraestructuras para Internet[9].

Avaya Inc. Es una empresa privada de telecomunicaciones que se especializa en el sector de la telefonía y centros de llamadas. Fue el proveedor oficial de comunicación convergente de la Copa Mundial de Clubes de la FIFA 2006. También proporcionó las redes de comunicaciones de la Copa Mundial de Fútbol 2002 y la Copa Mundial Femenina de la FIFA en 2003[10].

6. JAVACC (JAVA COMPILER COMPILER-METACOMPILADOR EN JAVA)

Es el principal metacompilador en Javacc, tanto por sus posibilidades, como por su ámbito de difusión. Se trata de una herramienta que facilita la construcción de analizadores léxico y sintáctico por el método de las funciones recursivas, aunque permite una notación muy relajada parecida a la BNF (abreviatura en inglés de Forma de Backus-Naur). De esta manera, los analizadores generados utilizan la técnica descendente a la hora de obtener el árbol sintáctico[11].

En 1996, Sun Microsystems liberó un parser llamado *Jack*. Los desarrolladores responsables de *Jack* crearon su propia compañía llamada *Metamata* y cambiaron el nombre *Jack* a JavaCC. *Metamata* se convirtió en *WebGain*. Después de que *WebGain* finalizara sus operaciones, JavaCC se trasladó a su ubicación actual[12].

Sus principales características son[13]

La estructura básica de un programa en Javacc es como aparece en la figura 3.

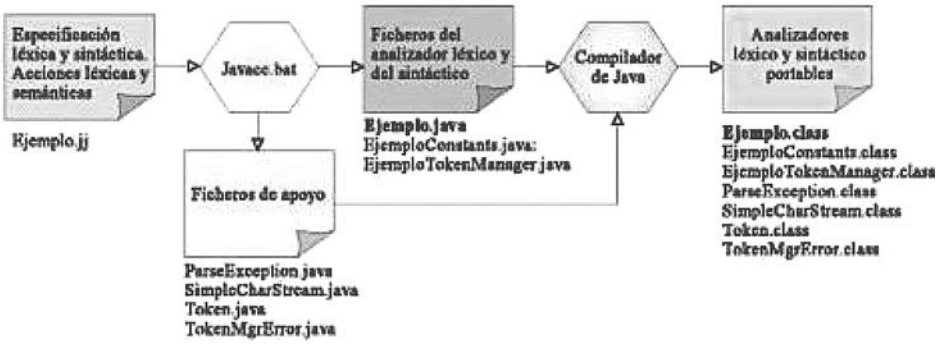
7. TRADUCCIÓN CISCO Y AVAYA

7.1. Comandos de cisco seleccionados para la traducción

Para efectos de traducción se escogió una muestra representativa de los comandos más importantes de CISCO que pueden ser traducidos a AVAYA. Estos comandos fueron probados inicialmente con un simulador para asegurar que las funciones realizadas por los comandos de las dos plataformas son iguales o similares. También se tuvo en cuenta el modo de operación de cada comando.

9. *Cisco systems en es.wikipedia.org/wiki/Cisco*
10. AVAYA en *wikipedia.org/wiki/Avaya*
11. GALVEZ, S. y MORA, M. A. (2004). Compiladores: Traductores y compiladores con Lex/yacc y Javacc. Universidad de Málaga. p. 131. Consultado en http://books.google.com.co/books?id=F3IWLs1iTAMC&printsec=frontcover#PPA131,M1
12. Consultado en http://es.wikipedia.org/wiki/JavaCC
13. GALVEZ, S. y MORA, M. A. (2004). Compiladores: Traductores y compiladores con Lex/yacc y Javacc. Universidad de Málaga. pp. 131-132. Consultado en http://books.google.com.co/books?id=F3IWLs1iTAMC&printsec=frontcover#PPA131,M1

Figura 3. Estructura de un programa en Javacc[14]



La siguiente es la lista de los comandos más representativos y susceptibles a ser traducidos. En esta tabla se presenta una columna con los comandos de CISCO a ser traducidos, una columna con los comandos de AVAYA hacia los cuales va dirigida la traducción y una columna de descripción que indica la función de cada comando y el modo de operación.

CISCO	AVAYA	DESCRIPCIÓN
Ping	Ping	Envía una petición de eco para diagnosticar la conectividad básica de red.
Show clock	Show time	Muestra la hora y fecha del equipo.
Show ip route	Show ip route	Muestra el contenido de la tabla de enrutamiento IP.
Exit	Exit	Salir de línea de comandos.
Copy tftp flash	Copy tftp module-config	Descarga una nueva imagen desde un servidor TFTP en la memoria Flash, en AVAYA la descarga a un módulo de los del dispositivo.
Show interfaces [tipo Número]	Show interface	Muestra estadísticas para la/las interfaces indicadas
Show arp	Show cam	Muestra la asignación de direcciones IP a MAC a interfaz del router.
Show interfaces	Show port	Muestra las interfaces - Puertos del dispositivo.
#Vlan database #no vlan [No de la Vlan]	Clear Vlan	Borra una Vlan.
Show vlan	Show vlan	Muetra una Vlan.
#Vlan database #vlan [No. de la Vlan] name [Nombre de la vlan]	Set vlan	Crea una Vlan en el dispositivo. Modo Cisco: modo vla

14. GALVEZ, S. y MORA, M. A. (2004). Compiladores: Traductores y compiladores con Lex/yacc y Javacc. Universidad de Malaga. p. 134. Consultado en http://books.google.com.co/books?id=F3lWLs1iTAMC&printsec=frontcover#PPA131,M1

CISCO	AVAYA	DESCRIPCIÓN
#interface [Tipo de interface] [Puerto No/Modulo] #switchport access vlan [No. Vlan]	Set port vlan	Asigna un puerto a una Vlan en particular. Modo Cisco: configuración terminal
#interface [Tipo de interface] [No. Puerto/Modulo] #switchport access vlan [1=Puerto por defecto]	Clear port static-vlan	Borra un puerto de una Vlan en particular. Modo Cisco: configuración terminal
#show spanning-tree	Show spantree	

8. METODOLOGÍA UTILIZADA EN LA TRADUCCIÓN

Para el reconocimiento de los comandos se utilizaron los lenguajes formales y gramática. Los lenguajes formales permiten identificar cada uno de los componentes o elementos individuales de cada comando. Mediante las expresiones regulares se especifican las reglas que permiten determinar la escritura de los comandos y la abreviación que lo representa así.

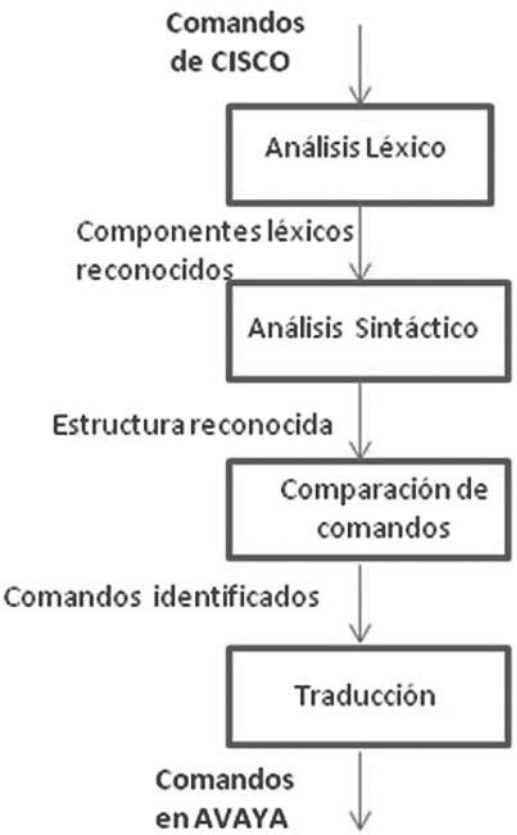
Si el usuario desea escribir en la plataforma CISCO el comando **Show clock**. Lo puede escribir en forma completa o abreviarlo así: **Sh cl** y el traductor lo reconocerá y realizará su respectiva traducción. También se tienen definidas todas las palabras reservadas.

Por otro lado, se utiliza la Gramática Independiente del Contexto cuando se desea reconocer una estructura sintáctica. Por ejemplo:

#Vlan database
#vlan [No. de la Vlan] name [Nombre de la vlan]

Para estos casos se realiza una Gramática independiente del Contexto que permite identificar y reconocer toda la estructura gramatical representada por la especificación de la plataforma CISCO. En la figura 4 se indica el procedimiento utilizado en la traducción.

Figura 4. Procedimiento de traducción comandos CISCO a AVAYA



Luego del reconocimiento léxico y sintáctico se procede a realizar la comparación de cada palabra reservada en CISCO con cada palabra reservada en AVAYA y por último se hace la traducción a los comandos en AVAYA.

9. RESULTADOS

9.1. Expresiones regulares y gramáticas indepedientes del contexto utilizadas en el traductor

Para el reconocimiento de componentes se utilizaron las expresiones regulares y para el reconocimiento de la estructura gramatical se utilizó una gramática independiente del contexto.

Ejemplos:
Comando Show clock. Las expresiones regulares en JavaCC son:

< SHOW: ("s"|"S")("h"|"H")(("o"|"O")(("w"|"W"))?)? > En esta expresión regular se determina la forma cómo se puede escribir *show*, la cual puede ser abreviada sh, sho o en forma completa show, también puede ser en mayúscula o minúscula.

<CLOCK: ("c"|"C")("l"|"L")(("o"|"O")(("c"|"C")(("k"|"K"))?)?)? > En esta expresión regular se determina cómo se puede escribir *clock* la cual puede ser abreviada cl, clo, cloc o simplemente en forma completa clock. También se controla que puede ser mayúscula o minúscula.

```
< EOL : "\n" | "\r" | "\r\n" > Final de linea
```

La gramática que permite establecer el orden lógico y significativo de los componentes es la siguiente:

```
void T() :
{}
{
 Y()T()
}
```

La gramática anterior equivale a la producción Y→YT donde Y y T son no terminales. Y y T se definen como sigue:

```
Void Y() :
{}
{
 <SHOW> ( B())
}
```

La gramática anterior equivale a la producción Y →SHOW B donde *show* es una expresión regular definida anteriormente y B es un no terminal definida como sigue:

```
void B() :
{}
{
 ( <CLOCK> ) (<EOL> |<EOF>)
 { System.out.println("Show clock");
}
}
```

B es un no terminal que puede ser CLOCK definida anteriormente o simplemente nada.

Otro ejemplo para la aplicación de las expresiones regulares y gramáticas es:

```
Vlan database
no vlan [No de la Vlan]
```

Expresiones regulares

```
<VLAN: ("v"|"V")("L"|"l")(("a"|"A")
(("n"|"N"))?)?> permite abreviar vl,
vla o simplemente vlan
<DATABASE:       ("d"|"D")(("a"|"A")
(("t"|"T")(("a"|"A")(("b"|"B")
(("a"|"A")(("s"|"S")
(("e"|"E"))?)?)?)?)?)?)?    >Permite
abreviar da, dat, data, datab, data-
ba, databas o simplemente la palabra
completa
<NO: ("n"|"N")("o"|"O") >
< EOL : "\n" | "\r" | "\r\n" > Final
de línea

< NUM: (["1"-"9"](<dig>)+ | ["2"-
"9"]) (" " | "\t" | "\n" | "\r") >
Números que empiezan con dígitos del
1 al 9 y continúan con cadenas de
dígitos.
| < ONE: "1" > Define al dígito 1.
```

Gramática Independiente del Contexto

```
void H() :
{}
{
 ( <VLAN> <DATABASE> ) (<EOL>)
( I() )
}
```

La no terminal H referencia la palabra reservada VLAN concatenada con DATABASE y luego se concatena con una terminal I que se define como sigue:

```
void I() :
{}
{
  ( <NO> <VLAN> ( <NUM> | <ONE> ) )
(<EOL> |<EOF>)
 { System.out.println("Clear Vlan");
}
}
```

La no terminal referencia las palabras reservadas NO VLAN seguida de NUM y ONE que son expresiones regulares definidas anteriormente.

9.2. Ejemplo del traductor

El traductor funciona de la siguiente forma. Inicialmente al entrar al programa se presenta la ventana principal que consta de dos áreas: área 1 (área izquierda) donde se ingresan los comandos en CISCO que se desean traducir, estos comandos pueden digitarse individualmente (ver figura 5) o colocarlos uno debajo del otro (ver figura 6). El área 2 (área de la derecha) contiene donde aparecerán los comandos en AVAYA producto de la traducción.

El traductor tiene la particularidad de reconocer e identificar los comandos de CISCO con solo escribir las iniciales de cada componente. Por ejemplo si desea traducir el comando **show clock**, solo puede digitar **sh cl** y él lo reconocerá e inmediatamente lo traducirá a su comando respectivo en AVAYA como se puede ver en la figura 7.

También el traductor está en capacidad de traducir comandos que ocupan más de una línea como se puede ver en la figura 8.

Figura 5. Traducción de un comando individual

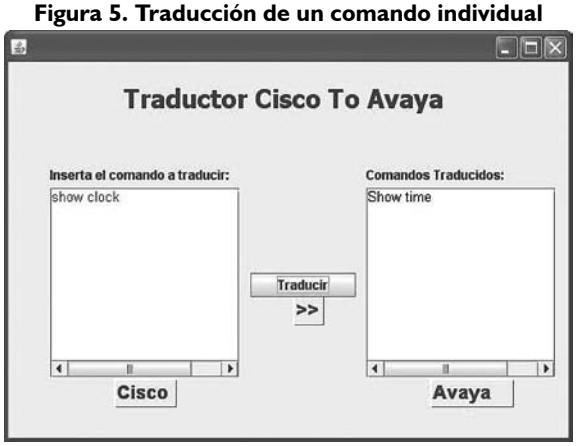


Figura 6. Traducción de varios comandos

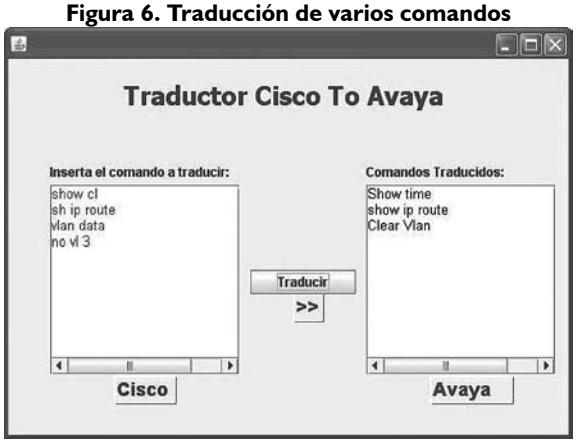


Figura 7. Simplificación de los comandos a traducir

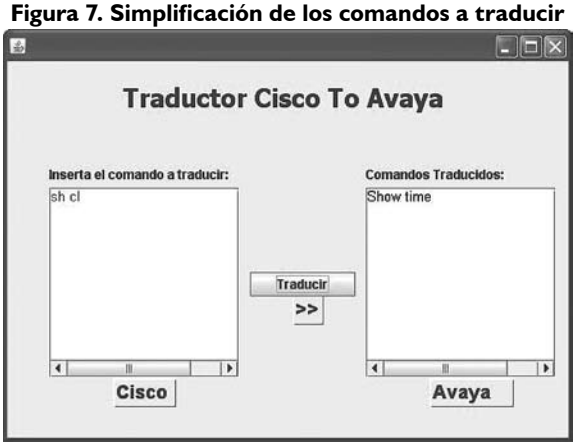
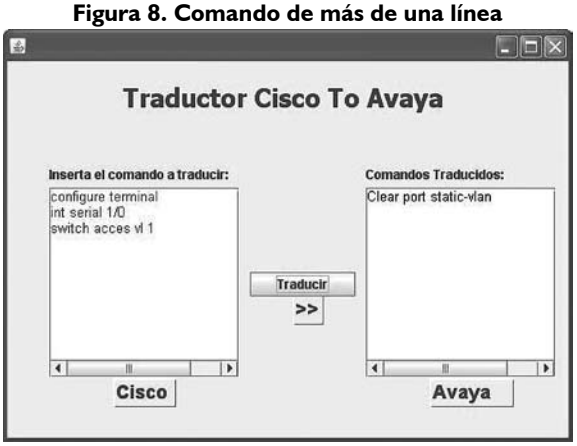


Figura 8. Comando de más de una línea



Es importante tener en cuenta que el traductor realiza un análisis léxico y sintáctico antes de realizar la traducción, de manera que cualquier error que se presente desde el punto léxico y sintáctico será reportado para su respectiva corrección. Hasta que no se realice la corrección no se podrá realizar la traducción. En la figura 9 se muestran los comandos a traducir y en la figura10 se muestran el mensaje de error indicando la posición muestra donde se produjo.

Figura 9. Comando de más de una línea

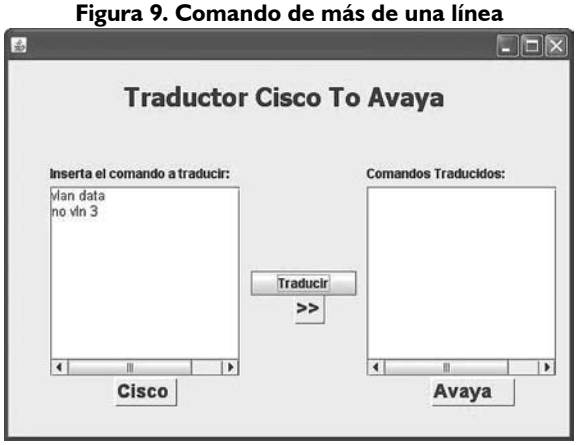
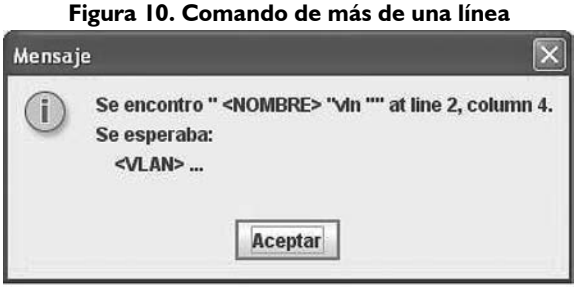


Figura 10. Comando de más de una línea



10. CONCLUSIONES Y TRABAJOS FUTUROS

Se espera que este proyecto genere un gran impacto sobre todo en los docentes y estudiantes de las asignaturas de Redes ya que será una herramienta de apoyo en el proceso de enseñanza-aprendizaje de la asignatura de Redes. Además será de gran importancia para el fortalecimiento de las capacidades científicas y tecnológicas de los estudiantes. También, sería un gran aporte para todos aquellos profesionales del área que realicen operaciones en una de las dos plataformas y quieran emigrar hacia la otra.

Por otro lado, mediante este software los estudiantes y profesionales podrían realizar comparaciones entre las dos plataformas. Si cuenta con las dos plataformas podría pasar los comandos de CISCO a AVAYA de forma rápida y segura, y si solo cuenta con la plataforma CISCO se utilizaría de forma didáctica para enseñar la analogía entre las dos plataformas.

De este trabajo se pueden plantear trabajos a corto y mediano plazo como crear un traductor de AVAYA a CISCO permitiendo de esta manera tener un traductor en dos vías. Este no sería un simple aporte didáctico sino también desde el punto de vista laboral ya que los ingenieros o personas encargadas de manejar las dos plataformas contarían con un software de soporte de gran ayuda en el proceso de reconocimiento y traducción de un comando a otro.

11. BIBLIOGRAFÍA

AHO, A. V.; SETHI, R. y ULLMAN, J. D. (1990) (Compiladores). *Principios, técnicas y herramientas*. Wilmington, Delaware: Addison-Wesley Iberoamericana S. A. pp. 1-5.

ARIGANELLO, E. Blog CCNA Aprende Redes en www.aprenderedes.com

BROOKSHEAR, J. G. y PEAKE, E. M. (1993). *Teoría de la computación: lenguajes formales, autómatas y complejidad*. Addison-Wesley. p. 110.

GALVEZ, S. y MORA, M. M. (2004, Mar). Compiladores. Traductores y compiladores con Lex/yacc y JavaCC. pp. 127-134. Disponible: http://books.google.com.co/books?id=F3lWLs1iTAMC&printsec=frontcover#PPA131,M1

KENNETH, L. C. (2004). *Construcción de compiladores. Principios y práctica*. México: Thomson Editores. pp. 31-35.

LEMONE, K. A. (1996). *Fundamentos de compiladores: Cómo traducir al lenguaje de computadora*. México: Compañía Editorial Continental S. A.

MARTÍN, J. (2004). *Lenguajes Formales y teoría de la Computación*. Editorial McGraw-Hill. pp. 12-25.

RODGER, S. H. y FINLEY, T. W. (2006, agosto). Tutorial de Jflap Disponible: http://www.jflap.org/tutorial/

RODRIGO, D. K. (2004). *Teoría de la Computación: lenguajes, autómatas y gramáticas*. Colombia: Unilibros. pp. 17-20.

SÁNCHEZ, G. y VALVERDE, J. A. (1989). Compiladores e intérpretes. *Un enfoque pragmático*. Ediciones Díaz de Santos. pp. 4-8.

TEUFEL, B.; SCHMIDT, S. y TEUFEL, T. (1993) (Compiladores). *Conceptos fundamentales*. Addison-Wesley Iberoamericana. pp. 10-15.

www.es.wikipedia.org/wiki/**JavaCC**

www.scribd.com/doc/99776/IntroduccionA**JavaCC**

www.geocities.com/raptware/**javacc**/index.html

www.cs.duke.edu/csed/**jflap**

http://es.wikipedia.org/wiki/Lenguaje_formal

Página Oficial JavaCC, en https://javacc.dev.java.net/

Documentación JavaCC en https://javacc.dev.java.net/doc/docindex.html

Tutorial JavaCC, en http://www.geocities.com/raptware/javacc/

http://www.javaworld.com/javaworld/jw-12-1996/jw-12-jack.html

http://www.javaworld.com/javaworld/jw-12-2000/jw-1229-cooltools.html

Expresiones regulares en Javacc en

http://www.geocities.com/raptware/javacc/Capitulo2.html

http://www.rpi.edu/

http://www.duke.edu/

http://trevinca.ei.uvigo.es/~formella/doc/talf05/talf/node42.html

*Cisco systems en es.wikipedia.org/wiki/**Cisco***