PROYECTO FIN DE GRADO

TÍTULO: Análisis y procesamiento de información textual utilizando generadores de compiladores

AUTOR/A: Miguel Amarís Martos

TITULACIÓN: Grado en Ingeniería Telemática

TUTOR/A: José Luis López Presa

DEPARTAMENTO: Ingeniería Telemática y Electrónica

VºBº TUTOR/A

Miembros del Tribunal Calificador:

PRESIDENTE/A: Rafael José Hernández Heredero

TUTOR/A: José Luis López Presa

SECRETARIO/A: Javier Malagón Hernández

Fecha de lectura: 10/07/2024

Calificación:

El Secretario/La Secretaria,

Resumen

En el panorama actual, el procesamiento de información en diversos formatos se ha convertido en una práctica común. Para abordar esta tarea, se emplean diferentes herramientas diseñadas específicamente para analizar datos en diversos lenguajes de programación o formatos como JSON o XML. Sin embargo, la necesidad de una solución genérica que permita el análisis de una amplia variedad de gramáticas se ha vuelto evidente. En este contexto, se han desarrollado generadores de reconocedores gramaticales, como Java Compiler Compiler (JavaCC). JavaCC es una herramienta que permite la generación de analizadores léxicos y sintácticos a partir de una gramática definida por el usuario. Su versatilidad y capacidad para adaptarse a diversas gramáticas lo convierten en una elección atractiva para el procesamiento de información estructurada. Este proyecto se enfoca en explorar el potencial de JavaCC y su aplicación en la asignatura "Procesamiento de la Información en Aplicaciones Telemáticas". El objetivo principal es aprender a utilizar JavaCC, aplicarlo en prácticas de PIAT y crear una documentación que respalde su versatilidad para el análisis de gramáticas y el procesamiento de información.

Abstract

In the current landscape, processing information in various formats has become a common practice. To address this task, different tools specifically designed to parse data in various programming languages or formats such as JSON or XML are used. However, the need for a generic solution that allows the parsing of a wide variety of grammars has become evident. In this context, grammar recogniser generators have been developed, such as Java Compiler Compiler (JavaCC). JavaCC is a tool that allows the generation of lexical and syntactic parsers from a user-defined grammar. Its versatility and ability to adapt to diverse grammars make it an attractive choice for structured information processing. This project focuses on exploring the potential of JavaCC and its application in the course "Information Processing in Telematics Applications". The main objective is to learn how to use JavaCC, apply it in PIAT practices and create documentation that supports its versatility for grammar parsing and information processing.

Agradecimientos

Quiero agradecer...

Índice de figuras

Índice general

Capítulo 1

Introducción

1.1. Contexto y motivación

El presente Proyecto Fin de Grado surge del interés personal por el procesamiento de información textual y la búsqueda de herramientas versátiles y eficientes para su análisis. Desde una temprana edad, me ha fascinado la capacidad del lenguaje para construir mundos, comunicar ideas y conectar con otras personas. En este sentido, la telemática me ha brindado la oportunidad de explorar el lenguaje desde una perspectiva más técnica y creativa, permitiéndome comprender cómo las máquinas pueden procesar y generar información textual.

A lo largo de mi formación académica, he tenido la oportunidad de aprender sobre diferentes herramientas para el procesamiento de información, como las clases de Java para formatos JSON o XML. Sin embargo, estas herramientas se limitan a un formato específico, lo que implica la necesidad de aprender un nuevo lenguaje para cada caso. Esta limitación me motivó a buscar una alternativa más universal que me permitiera abordar cualquier tipo de formato textual sin necesidad de un aprendizaje adicional.

En este contexto, descubrí JavaCC, un generador de analizadores léxicos y sintácticos que me fascinó por su potencia y flexibilidad[1]. Esta herramienta me abrió un mundo de posibilidades al permitirme concebir una forma de procesar información de forma genérica, independientemente del formato en el que se presente.

Este proyecto se presenta como una oportunidad excepcional para profundizar en mi conocimiento del lenguaje y explorar sus aplicaciones en el ámbito del procesamiento de información. Además, me motiva la idea de contribuir a la asignatura de PIAT —perteneciente al Grado en Ingeniería Telemática de la Escuela Técnica Superior de Sistemas de Telecomunicación— con una documentación completa y accesible que facilite el aprendizaje de esta herramienta a otros estudiantes y profesorado.

Más allá de las ventajas técnicas, este proyecto me permite conectar con mi pasión por el lenguaje y explorar su potencial en el mundo digital. Estoy convencido de que este proyecto le aportará al lector conocimientos técnicos valiosos, además de habilitarle a desarrollar sus capacidades de aprendizaje autónomo, investigación y creatividad.

En definitiva, este proyecto representa una oportunidad única para combinar mi pasión por el lenguaje con mi formación en telemática, permitiéndome contribuir al desarrollo de nuevas herramientas y conocimientos en el ámbito del procesamiento de información textual.

1.2. Objetivos

Como se acaba de explicar, el objetivo principal del proyecto es evaluar la viabilidad a la hora de utilizar una herramienta de generadores genéricos como JavaCC para abordar las prácticas de PIAT.

Esto implica aprender a utilizar la herramienta JavaCC de manera efectiva. En otras palabras, ahondaremos en las bases fundamentales de los analizadores y comprenderemos las virtudes y los defectos de este tipo de herramientas. Acto seguido, aplicaremos los conocimientos adquiridos al estudiar la herramienta JavaCC en prácticas específicas de la asignatura PIAT. Observaremos las carencias y limitaciones que lamentablemente se enseñan en dichos ejercicios para ofrecer una solución elegante, sencilla y extremadamente potente empleando la herramienta JavaCC.

Por último, el objetivo final de dicho proyecto es el de generar una documentación que sirva como guía de uso para estudiantes y profesores interesados en utilizar JavaCC en proyectos relacionados con el procesamiento de información.

En definitiva, se busca proporcionar una solución versátil y eficaz para el análisis de gramáticas y el procesamiento de información en el ámbito de prácticas de PIAT.

1.3. Restricciones

A la hora de desarrollar y realizar del proyecto, debemos tener en cuenta las siguientes restricciones:

1. El proyecto se centrará en la utilización de JavaCC como herramienta principal.
2. Debe garantizarse que la documentación generada de dicha herramienta sea comprensible y útil para aquellos que deseen aplicar JavaCC en proyectos similares.
3. Las prácticas y ejercicios de la asignatura PIAT deben servir como un contexto relevante para la aplicación de JavaCC.

Estas restricciones sirven como nuestro marco de referencia, proporcionando directrices claras sobre cómo se llevará a cabo el proyecto y asegurando que nos mantenemos fieles a nuestra visión original. Son los pilares que sostienen nuestra estrategia de proyecto y nos ayudan a mantener un enfoque claro y coherente.

A medida que avanzamos en el proyecto, estos principios y objetivos nos servirán de guía, asegurando que permanecemos en el camino correcto y que cada paso que damos nos acerca a nuestra meta final. Cada decisión que tomamos, cada desafío que enfrentamos, se evalúa en función de estas restricciones. De esta manera, nos aseguramos de que nuestro proyecto no solo cumple con los requisitos establecidos, sino que también aporta valor y conocimiento al campo de los analizadores de lenguaje.

1.4. Estructura del documento

Esta sección proporciona un esquema detallado de la estructura adoptada para la ejecución de este proyecto.

Comenzamos con el *Capítulo 1. Introducción.* Este capítulo sirve como punto de partida para el proyecto, estableciendo el escenario para lo que está por venir.

Aquí se articula la razón detrás de la concepción del proyecto, que se deriva tanto del interés personal como de la relevancia que este proyecto tendrá en el campo. También se delinean los objetivos tangibles, en términos de los resultados que esperamos producir, como intangibles, en términos del conocimiento y la experiencia que esperamos adquirir en el proceso. Además, se establecen las bases y principios que guiarán este proyecto.

Una vez concluida esta sección, nos adentramos en el *Capítulo 2. Estado del arte*, en el cual exploraremos las tecnologías empleadas en la actualidad en el ámbito de los analizadores de lenguaje. Además, nos familiarizaremos con una variedad de términos técnicos y jerga sintáctica que se utilizan en este campo. Esto proporcionará al lector una visión completa de las herramientas y técnicas actuales, permitiendo una comprensión más profunda de cómo funcionan los analizadores de lenguaje y cómo se pueden aplicar en diferentes contextos. También se discutirán las ventajas y desventajas de estas tecnologías, y se presentarán tanto ejemplos prácticos como las tendencias emergentes.

El *Capítulo 3. Desarrollo* se dedica a la exploración en profundidad de los elementos relacionados con la implementación del proyecto. En él se detallarán el proceso de implementación, los procedimientos seguidos, el entorno de desarrollo y las pruebas, y la realización, estudio y desarrollo de las prácticas de PIAT utilizando JavaCC.

Una vez realizados los desarrollos e implementaciones correspondientes, en el *Capítulo 4. Resultados*, se presentarán y analizarán los resultados obtenidos a lo largo del desarrollo del proyecto. Se revelarán los resultados en términos de los objetivos establecidos al inicio del proyecto, se escudriñarán los resultados en términos de la eficacia de las técnicas y metodologías utilizadas, y se discutirá los resultados en términos de la calidad del producto final revelando las lecciones aprendidas durante el desarrollo del proyecto.

Todos estos apartados convergerán en el *Capítulo 5, Conclusiones*, en el cual se ofrecerá una revisión general del proyecto, presentando las conclusiones extraídas, reflexionando sobre el proceso de desarrollo y discutiendo las posibilidades de trabajo futuro.

Finalmente, el *Apéndice* proporciona información adicional, incluyendo detalles técnicos, documentación detallada, código fuente y otros datos relevantes. Estos detalles pueden ser útiles para aquellos lectores que deseen replicar el entorno de desarrollo o entender mejor cómo se configuró y utilizó cada herramienta. Además, se incluye el código fuente de partes significativas del desarrollo, así como cualquier información relevante pero que no se ajusta a los capítulos principales.

Capítulo 2

Estado del arte

2.1. Marco tecnológico

El procesamiento de información en diversos formatos es una tarea cada vez más importante. En un mundo cada vez más digital, es necesario poder procesar datos en una amplia variedad de formatos, desde lenguajes de programación hasta formatos de documentos y datos estructurados. En este contexto, los generadores de reconocedores gramaticales, como JavaCC, ofrecen una solución muy prometedora. Estos generadores permiten crear analizadores léxicos y sintácticos a partir de una gramática definida por el usuario. Esto los hace muy versátiles y adaptables a una amplia variedad de gramáticas. La aplicación de JavaCC en la asignatura Procesamiento de la Información en Aplicaciones Telemáticas (PIAT) es una idea muy acertada. PIAT es una asignatura que se centra en el procesamiento de información en diversos formatos. El uso de JavaCC podría proporcionar a los estudiantes una herramienta muy valiosa para abordar las prácticas de la asignatura.

2.2. Analizador Léxico

En el contexto de los analizadores de compiladores, un **analizador léxico** (del griego `lexis`, palabra) —también conocido como *lexer*— es la **primera fase** del proceso de **compilación**. Su función es analizar el código fuente de un programa escrito en un lenguaje de programación y producir una secuencia de tokens o componentes léxicos[2]. Con esto conseguimos reconocer tokens —hablaremos de ellos más adelante–, ya que el analizador léxico divide la sintaxis del programa en una serie de tokens —palabras clave, identificadores, operadores...— y es capaz de eliminar cualquier información no relevante para el análisis posterior, como espacios adicionales o comentarios.

Adicionalmente, si encuentra tokens inválidos o mal formados, el analizador léxico tiene la capacidad de generar mensajes de error. En última instancia, el analizador léxico **se encarga de preparar la entrada para** el siguiente paso del proceso de compilación, que es **el análisis sintáctico**.

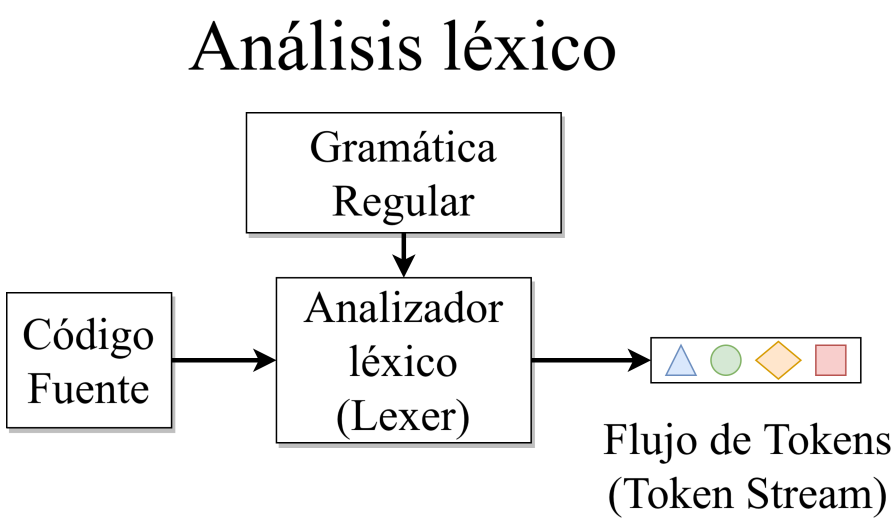

Figura 1: Funcionamiento del Analizador Léxico en JavaCC[3]

Supongamos que tenemos la siguiente frase: `How Pleasant Is The Weather?` . Aquí podemos reconocer fácilmente que hay cinco palabras: "*How*", "*Pleasant*", "*Is*", "*The*" y "*Weather*." Esto es natural para nosotros, ya que podemos identificar los separadores (espacios en blanco) y el símbolo de puntuación.

Ahora, consideremos una variante de la misma frase: `HowPl easantIs Th ewe ather?`. Aquí es donde entra en juego el analizador léxico. Aunque también podemos leer esta versión, llevará más tiempo porque los separadores se colocan en lugares impares. No es algo que se entienda de inmediato.

El analizador léxico escanea el código fuente y reconoce los tokens incluso en situaciones más complejas como esta. En este caso, identificaría las palabras como tokens individuales, a pesar de la falta de espacios.

A continuación se va a ilustrar otro ejemplo, mas orientado a analizar archivos de código. Supongamos un fragmento de código en en un lenguaje de programación ficticio como este:

```
suma = 10 + 20
resta = 30 - 15
multiplicacion = 5 * 6
division = 24 / 4
```

En este caso, el analizador léxico se encargaría de escanear este código fuente y dividirlo en tokens significativos. El analizador léxico reconocería las palabras clave como `suma`, `resta`, `multiplicacion`, y `division`. Además, también identificaría los operadores aritméticos como `+`, `-`, `*`, y `/`, además de números enteros como `10`, `20`, `30`, `15`, `5`, y `6`. Por otra parte, El analizador léxico eliminaría cualquier espacio en blanco o comentario presente en el código. El resultado sería una secuencia de tokens como sigue:

```
suma,=,10,+,20,resta,=,30,-,15,multiplicacion,=,5,*,6,division
   ,=,24,/,4
```

2.3. Token

Un token es una unidad léxica o componente básico del código fuente de un programa. Se trata de una cadena de caracteres con un significado específico asignado. Está estructurado como un par que consta de un nombre de token y un valor de token opcional. El nombre del token representa una categoría o tipo de unidad léxica, como palabras clave, identificadores, operadores, entre otros[4].

En el ejemplo de la sección anterior, el analizador léxico generaba una secuencia de tokens, cada token tenía sus características propias, dependiendo de si eran identificadores, operadores o signos de puntuación, u otros —se pueden especificar token que hagan referencias a comentarios dentro del código, o palabras reservadas del lenguaje, como `if` o `while`—.

2.4. Analizador Sintáctico o Semántico

Un **analizador sintáctico** es una parte fundamental de un **compilador** o intérprete. Su función es verificar si un programa fuente —escrito en un lenguaje de programación— sigue las reglas gramaticales definidas para ese lenguaje[5]. En otras palabras, nos indica la disposición correcta de los elementos en el código fuente para que se convierta en un programa válido.

Comúnmente conocido como *parser*, el analizador sintáctico verifica la **estructura** del programa fuente. Este hace uso de una gramática (generalmente una gramática libre de contexto) para definir las reglas sintácticas del lenguaje. El objetivo del parser es reconocer la secuencia de *tokens* (unidades léxicas como palabras clave, identificadores, operadores) y construir un **árbol sintáctico** que represente la estructura jerárquica del programa. Si el archivo fuente es válido, el analizador sintáctico proporciona el árbol sintáctico correspondiente.

Además de todas las funciones mencionadas, si encuentra errores sintácticos, como paréntesis desequilibrados o expresiones mal formadas, el analizador sintáctico genera mensajes de error[6]. De esta forma ayuda a los programadores a identificar y corregir problemas en el código fuente de una manera sencilla y muy efectiva. Otras funciones que los analizadores sintácticos pueden desempeñar son el acceder a la tabla de símbolos, ya que necesita información sobre variables, funciones y otros símbolos definidos en el programa; el chequeo de tipos, ya que a menudo se verifica la compatibilidad de tipos —por ejemplo, si se está aplicando un operador a operandos compatibles— ; y la generación de código intermedio, proporcionando así una representación más abstracta y simplificada del programa.

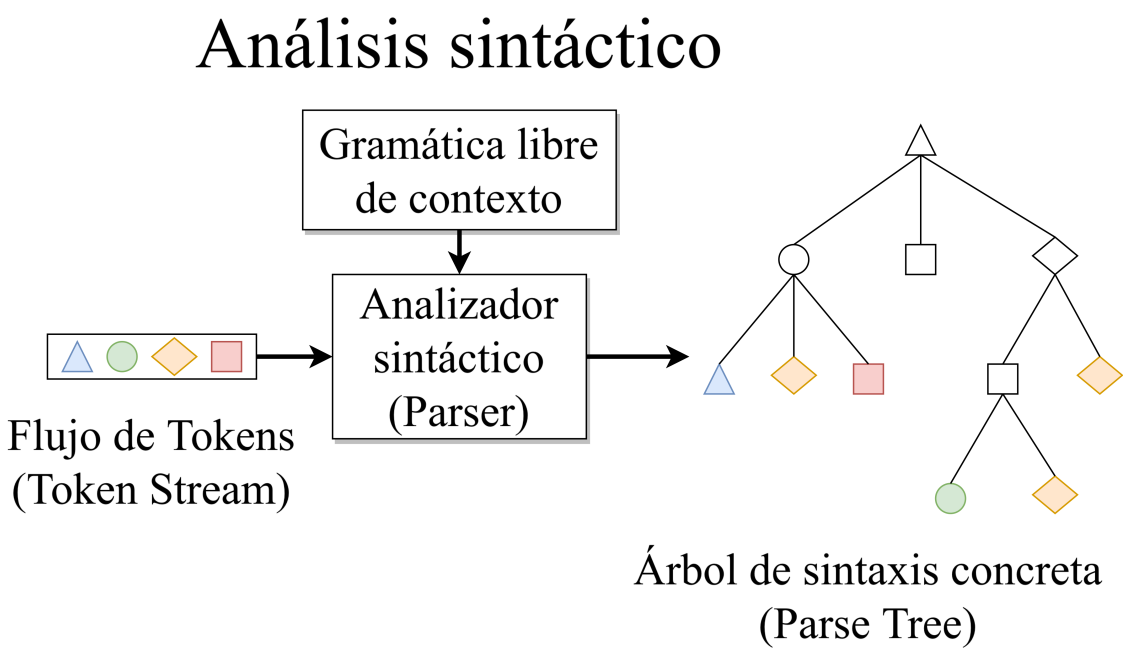

Figura 2: Funcionamiento del Analizador Sintáctico en JavaCC[7]

2.5. Estados Léxicos

Los estados léxicos permiten efectuar un conjunto diferente de producciones de expresiones regulares en un caso concreto. Dicho de otra manera, los estados léxicos sirven para que un token este definido de una forma, y en cierto momento del análisis, pueda estar definido de otra forma.

Supongamos que desea escribir un procesador JavaDoc para extraer los comentarios JavaDoc de un documento. La mayor parte de Java está tokenizada de acuerdo con las reglas ordinarias regulares de Java. Pero dentro de los comentarios JavaDoc, se aplica un conjunto diferente de reglas en las que las palabras clave deben ser reconocidas y donde las nuevas líneas son significativas —como las palabras y símbolos `/***/` , `@param`, entre otros—.

Para resolver este problema, podríamos usar dos estados léxicos: uno para la tokenización regular de Java y otro para la tokenización dentro de los comentarios de JavaDoc.

2.6. Recursividad

En el ámbito de los compiladores, la recursividad es una técnica que se utiliza para definir una estructura gramatical que se repite en sí misma.

Esto permite resolver problemas complejos dividiéndolos en subproblemas más pequeños. Este proceso se repite hasta que se llega a subproblemas que son triviales de resolver.

En JavaCC, la recursividad se utiliza para definir producciones gramaticales que pueden analizar expresiones complejas.

2.6.1. Recursividad a derechas (LR)

En la recursividad a derechas, la llamada recursiva se realiza al final de la definición de la estructura gramatical. Por ejemplo, la gramática para la expresión aritmética básica se puede definir recursivamente a la derecha de la siguiente manera:

```
Expression() :
{
  Token t;
}
{
  "(" e = Expression() ")"
  {
    return e;
  }
}
```

2.6.2. Recursividad a izquierdas (LL)

En la recursividad a izquierdas, la llamada recursiva se realiza al principio de la definición de la estructura gramatical. Por ejemplo, la gramática para la expresión aritmética básica se puede definir recursivamente a la izquierda de la siguiente manera:

```
Expression() :
{
  double e;
}
{
  e = Term()
  (
    "+" e = Term()
  |
    "-" e = Term()
  )*
  {
    return e;
  }
}
```

2.6.3. ¿Por qué no se puede utilizar la recursividad a izquierdas en JavaCC?

La clase de analizador sintáctico producida por JavaCC funciona por descenso recursivo. La recursión a la izquierda está prohibida para evitar que las subrutinas generadas se llamen a sí mismas recursivamente de forma infinita.

Si considerásemos una producción recursiva a la izquierda, en caso de que la condición fuera verdadera alguna vez, tendríamos una recursión infinita. JavaCC producirá un mensaje de error si tiene producciones recursivas a la izquierda.

2.7. JavaCC

En relación con lo anterior, aclaremos a qué hace referencia el concepto de JavaCC y los analizadores semánticos y sintácticos, y como se interrelacionan: Java Compiler Compiler, comúnmente conocido como JavaCC, es una poderosa herramienta ampliamente utilizada para generar analizadores sintácticos en aplicaciones Java. Su función principal es transformar una especificación gramatical en un programa Java capaz de reconocer y analizar la sintaxis según la gramática proporcionada. JavaCC se diferencia de otras herramientas similares al generar analizadores sintácticos de arriba hacia abajo (descenso recursivo) en lugar de analizadores de abajo hacia arriba. Esta característica permite el uso de gramáticas más generales y facilita la depuración, además de permitir el análisis en cualquier elemento no terminal de la gramática y la transferencia de valores (atributos) en ambas direcciones en el árbol de análisis.

2.8. Funciones principales de JavaCC

JavaCC ofrece una serie de características y funcionalidades clave que lo hacen destacar como un generador de analizadores sintácticos:

- Generación de analizadores sintácticos de arriba hacia abajo.
- Resolución de ambigüedades de cambio de turno localmente.
- Generación de analizadores 100 % Java puro.
- Especificaciones BNF extendidas.
- Integración de especificaciones léxicas y gramaticales en un solo archivo.
- Manejo de la entrada Unicode completa.
- Soporte para tokens no distinguibles entre mayúsculas y minúsculas.
- Herramientas adicionales como JJTree para construir árboles y JJDoc para generar documentación.
- Personalización a través de numerosas opciones.
- Informes de errores de alta calidad y mensajes de diagnóstico completos.

2.9. Ejemplo de gramática en JavaCC

A continuación, se presenta un ejemplo de cómo JavaCC puede utilizarse para generar un analizador sintáctico en Java. En este ejemplo se define un lenguaje para analizar expresiones matemáticas simples. Partiendo de este ejemplo, seremos capaz de ampliar las funcionalidades del programa, y realizar una calculadora. El resultado final se encuentra en el archivo Mathexp.jj, que se encuentra en el apéndice *Código fuente*.

Función Expression()

```
void Expression() : {}
{
    Term() ( "+" Term() | "-" Term() )*
```

```
}
```

La función *Expression()* define la estructura general de una expresión. Una expresión consiste en un término seguido de cero o más términos conectados por operadores aritméticos. La función Expression() comienza con la función Term(). Esto significa que la primera parte de cualquier expresión debe ser un término. A continuación, la función Expression() utiliza la regla)* para especificar que puede seguir cero o más términos. Esto significa que las expresiones pueden tener cualquier longitud, desde una sola palabra hasta una expresión compleja con muchos términos. Los términos están conectados por operadores aritméticos. La regla Expression() utiliza la regla)* para especificar que puede seguir cualquier número de operadores aritméticos. Esto significa que las expresiones pueden tener cualquier cantidad de operadores aritméticos, desde ninguno hasta muchos. Para más información acerca de operadores en JavaCC, vaya al anexo Operadores en JavaCC. Los operadores aritméticos permitidos son la suma (+), la resta (-), la multiplicación (*) y la división (/).

Función Term()

```
void Term() : {}
{
    Factor() ( "*" Factor() | "/" Factor() )*
}
```

La regla *Term()* define la estructura de un término. Un término consiste en un factor seguido de cero o más factores conectados por operadores aritméticos. La regla *Term()* comienza con la regla *Factor()*. Esto significa que la primera parte de cualquier término debe ser un factor. A continuación, la regla *Term()* utiliza la regla `)*` para especificar que puede seguir cero o más factores. Esto significa que los términos pueden tener cualquier longitud, desde una sola palabra hasta un término complejo con muchos factores. Los factores están conectados por operadores aritméticos. La regla *Term()* utiliza la regla `)*` para especificar que puede seguir cualquier número de operadores aritméticos. Esto significa que los términos pueden tener cualquier cantidad de operadores aritméticos, desde ninguno hasta muchos. Los operadores aritméticos permitidos son la suma `+`, la resta `-`, la multiplicación `*` y la división `/`

Función Factor()

```
void Factor() : {}
{
    <NUMBER>
    | "(" Expression() ")"
}
```

La regla *Factor()* define la estructura de un factor. Un factor puede ser un número o una expresión entre paréntesis. La regla *Factor()* tiene dos opciones. La primera opción es un número. La segunda opción es una expresión entre paréntesis. Si la opción elegida es un número, la regla *Factor()* utiliza la regla `<NUMBER>` para especificar que el factor debe ser un número. Si la opción elegida es una expresión entre paréntesis, la regla Factor() utiliza la regla "(" *Expression()* ")" para especificar

que la expresión debe estar entre paréntesis.

Ejemplos

Aquí hay algunos ejemplos de expresiones que pueden ser analizadas por esta gramática:

```
1 + 2
3 * 4
(5 - 6) / 7
```

Aunque también se puede analizar expresiones más complejas, como:

```
(1 + 2) * (3 - 4)
(5 * 6) / (7 + 8)
```

Ampliación de la funcionalidad a Calculadora

A continuación, vamos a ampliar las capacidades de nuestro programa, para analizar las expresiones y hacer las operaciones de una calculadora: NL_Xlator.jj

Esta clase traduce expresiones matemáticas válidas en sus valores numéricos correspondientes. El usuario puede ingresar las expresiones matemáticas una por una, separándolas por punto y coma. La clase lee las expresiones del flujo de entrada estándar y las traduce a sus valores numéricos correspondientes. El resultado de la traducción se muestra en la salida estándar.

Análisis sintáctico

La clase utiliza JavaCC para analizar las expresiones matemáticas ingresadas por el usuario. JavaCC es una herramienta de generación de analizadores de sintaxis que permite crear analizadores personalizados para lenguajes de programación o lenguajes de dominio específicos.

Expresión raíz

La expresión raíz de la gramática es *ExpressionList()*. Esta producción analiza una lista de expresiones matemáticas separadas por punto y coma. Cada expresión matemática es analizada por la producción Expression().

Expresión

La producción *Expression()* analiza una expresión matemática completa. Esta producción analiza un término (Term()) seguido de cero o más operadores de suma o resta (+ o -) y términos.

Término

La producción *Term()* analiza un término matemático. Esta producción analiza un factor (*Factor()*) seguido de cero o más operadores de multiplicación o división (* o /) y factores.

Factor

La producción *Factor()* analiza un factor matemático. Esta producción analiza un número (NUM) o un par de paréntesis ((,)) que encierran una expresión matemática.

Errores

La clase también maneja errores sintácticos. Cuando ocurre un error sintáctico, la clase muestra un mensaje de error en la salida estándar y termina la ejecución.

Ejemplo de uso

Para utilizar la clase, simplemente ejecute el archivo NL_Xlator.java. A continuación, se le pedirá que ingrese expresiones matemáticas una por una, separándolas por punto y coma. La clase le mostrará el resultado de la traducción de cada expresión. Aquí hay un ejemplo de cómo utilizar la clase:

2.10. Usos de JavaCC en la actualidad

JavaCC es una herramienta muy completa a la hora de construir de analizadores léxicos y sintácticos para lenguajes de programación. Si bien su desarrollo original se centró en Java, su flexibilidad y robustez lo han convertido en una herramienta versátil con aplicaciones en diversos campos.

En el contexto de los lenguajes de programación, JavaCC ha tenido varias aplicaciones, como es el caso de Apache. A modo de ejemplo del proyecto Apache, JavaCC se utiliza en el proyecto Apache Ant para analizar archivos `build.xml`, que definen las tareas de construcción de software[8], aunque indagaremos en unos momentos en el proyecto Apache. Por otro lado, JavaCC se ha utilizado para construir compiladores e intérpretes para diversos lenguajes de programación, como Python, Ruby, C#, PHP, y JavaScript[9].

Otra aplicación en la que JavaCC se puede desenvolver con comodidad es el análisis del lenguaje natural. JavaCC se utiliza para analizar la estructura sintáctica de oraciones en aplicaciones de procesamiento de lenguaje natural[10]. También se utiliza para extraer información de documentos de texto, como nombres de entidades, fechas y lugares.

Si nos centramos en el marco actual, hay un gran repertorio de aplicaciones comerciales que hacen uso de JavaCC. Un buen ejemplo de ello es el proyecto de Apache.

Apache

Apache Software Foundation es una comunidad descentralizada de desarrolladores que trabajan en sus propios proyectos de código abierto. Los proyectos Apache se caracterizan por un modelo de desarrollo basado en el consenso, la colaboración y una licencia de software abierta y pragmática[11]. Uno de los proyectos notables dentro de la Apache Software Foundation es el servidor web Apache HTTP, que consiste en un servidor web HTTP de código abierto utilizado para crear páginas y servicios web. Es multiplataforma, gratuito, robusto y se destaca por su seguridad y rendimiento[12].

Entre los proyectos Apache en los que JavaCC se utiliza, cabe destacar Apache Lucene — en el que se emplea para procesar consultas, permitiendo interpretar y procesar estructuras gramaticales definidas en lenguajes específicos—, Apache Avro —que trasforma lenguajes de alto nivel a esquemas Avro— , o por ejemplo Apache Tomcat —que parsea expresiones de lenguaje (*Expression Language*)[13] y JSON— [1].

Como se ha mencionado antes, en el proyecto de Apache Ant se utiliza JavaCC para analizar archivos `build.xml`. JavaCC se encarga de analizar la sintaxis del archivo y construir un árbol de sintaxis, mientras que Ant coordina y automatiza las tareas de construcción y despliegue.

JavaCC también se utiliza en el parseo de archivos desarrollados en Java, como puede ser la librería JavaParser[14]. Otros proyectos en los que se utilice JavaCC, aparte de Apache, son JFlex y ANTLR.

JFlex

JFlex es una herramienta para la construcción de analizadores léxicos. Este está escrito en Java y se utiliza junto con JavaCC para analizar la sintaxis de las reglas léxicas. Por una parte, JFlex se encarga de generar el analizador léxico que escanea el código fuente y produce tokens. Posteriormente, se utiliza JavaCC para generar el analizador sintáctico que procesa los tokens generados por el escáner léxico y aplica las reglas gramaticales

ANTLR

ANTLR (*ANother Tool for Language Recognition*) es una herramienta para la construcción de analizadores léxicos y sintácticos. En términos de funcionalidades, es una herramienta más completa que abarca tanto el análisis léxico como el sintáctico. Sin embargo, JavaCC es más rápido y más fácil de aprender para un desarrollador Java, ya que la sintaxis es muy parecida, ademas de que ofrece una buena integración con IDEs[15]. Además de generar analizadores sintácticos, también crea analizadores léxicos (escáneres) y árboles de sintaxis abstracta (AST). ANTLR está escrito en Java y se puede utilizar junto con JavaCC para analizar la sintaxis de las gramáticas.

2.11. Procesamiento de la Información en Aplicaciones Telemáticas (PIAT)

2.11.1. Introducción

Las prácticas de PIAT son una serie de ejercicios que se utilizan para enseñar a los estudiantes de la asignatura los fundamentos de la programación orientada a objetos, y como se puede utilizar para analizar y extraer información de distintos formatos. Estas prácticas incluyen ejercicios de análisis de ficheros XML y JSON, principalmente.

La aplicación de JavaCC en las prácticas de PIAT tiene varias ventajas potenciales. En primer lugar, puede ayudar a los estudiantes a aprender los fundamentos de la programación orientada a objetos de una manera más eficiente y de una forma que todavía no han tenido la oportunidad de aprender y explotar. En segundo lugar, puede ayudar a los estudiantes a desarrollar sus habilidades de programación, y por consecuencia, ayudar a los estudiantes a crear código más eficiente y de mejor calidad.

2.11.2. Problemática con las prácticas de PIAT

Las prácticas de PIAT, que se centran en el análisis de archivos XML o JSON, utilizan soluciones como SAXParser o XPath —las veremos en detalle mas adelante—. Estas soluciones, sin embargo, presentan una serie de problemas que pueden dificultar el desarrollo de código eficiente y mantenible.

Uno de los principales problemas de SAXParser y XPath es que generan mucho código repetitivo. Los bucles while y loops que se utilizan para recorrer el árbol de datos pueden ser muy largos y complejos, lo que dificulta la comprensión y el mantenimiento del código.

Otro problema de estas soluciones es que pueden ser ineficientes en el rendimiento. SAXParser, por ejemplo, se basa en un modelo de eventos, lo que significa que el código debe estar preparado para procesar cualquier tipo de evento que pueda ocurrir. Esto puede provocar que el código se ejecute de forma innecesaria, lo que puede afectar al rendimiento de la aplicación.

Una buena solución a este tipo de soluciones es un generador de analizadores sintácticos como JavaCC. JavaCC permite crear analizadores sintácticos a partir de una gramática definida por el usuario. Esto permite generar código más eficiente y fácil de mantener, ya que el analizador sintáctico se genera automáticamente a partir de la gramática.

Al definir la gramática del analizador sintáctico de forma personalizada, este ofrece una gran flexibilidad, ya que permite adaptar el analizador a las necesidades específicas de la aplicación. Por ejemplo, si queremos analizar un archivo XML que tiene una estructura específica, podemos definir una gramática que refleje esa estructura. Esto nos permitirá generar un analizador que sea más eficiente y fácil de mantener.

Además, una vez que se ha definido la gramática, JavaCC genera el código fuente del analizador sintáctico. Este código fuente se puede modificar a conveniencia, lo que nos permite adaptar el analizador a las necesidades específicas de la aplicación. Esto permite que, si queremos añadir nuevas funcionalidades al analizador, podemos hacerlo fácilmente modificando el código fuente. Esto nos permite tener un mayor control sobre el analizador y adaptarlo a las necesidades cambiantes de la aplicación, característica clave en el entorno de las practicas de PIAT, ya que se caracterizan por tener un estilo incremental y con modificaciones a lo largo de cada ejercicio.

Por otra parte, si bien es cierto que JavaCC puede ser una herramienta compleja para aprender a utilizar, ya que requiere un conocimiento básico de gramáticas y de la sintaxis de Java, el potencial que ofrece JavaCC para desarrollar analizadores sintácticos eficientes y adaptables es muy alto.

Una vez se ha aprendido a utilizar JavaCC, se puede utilizar para desarrollar analizadores sintácticos para una amplia gama de lenguajes, incluyendo XML, JSON, HTML, C++, PHP, Cobol, SQL, IDL, entre otros[1].

Capítulo 3

Desarrollo

3.1. Introducción

En el capítulo anterior se han expuesto las principales tecnologías que forman parte del proyecto de evaluación de la viabilidad de JavaCC para el procesamiento de información en la asignatura PIAT. En este capítulo se desarrollarán los aspectos relativos a la implementación del proyecto, los procedimientos seguidos y el entorno utilizado para los desarrollos y las pruebas.

3.2. Implementación

La implementación del proyecto se ha realizado en Java, utilizando la herramienta JavaCC para la generación de los analizadores léxico y sintáctico. Para la creación de las gramáticas se ha utilizado el lenguaje EBNF.

3.3. Procedimientos

El proceso de implementación se ha dividido en las siguientes fases:

- Fase 1: Aprendizaje de los conceptos básicos de JavaCC.
- Fase 2: Práctica con JavaCC para crear analizadores léxicos y sintácticos para gramáticas simples.
- Fase 3: Aplicación de JavaCC en prácticas de PIAT.
- Fase 4: Generación de documentación.

3.4. Entorno

Los desarrollos y las pruebas se han realizado en un entorno de desarrollo integrado (IDE) de Java. En concreto, se ha empleado Eclipse IDE ya que, además de ser el IDE principal con el que se realizan las prácticas de PIAT —y en general, cualquier práctica relacionada con la programación dentro de la escuela—, existe un plugin de JavaCC que realiza la compilación de los archivos y facilita enormemente el desarrollo. Dicho esto, el lector puede optar por elegir otros editores como Visual Studio Code o Jetbrains, incluso editores de texto como Notepad++, Vim o nano. Este aspecto queda a gusto del desarrollador. Para la depuración de los analizadores se ha utilizado el debugger del IDE.

Para saber mas acerca de la instalación y configuración de Elipse IDE con JavaCC, puede consultar el anexo *Instalación de JavaCC*.

3.5. Objetivos de la implementación

Entre los objetivos principales de la implementación se encuentran el implementar los analizadores léxico y sintáctico para las gramáticas de las prácticas de PIAT, el evaluar la viabilidad de utilizar JavaCC para abordar las prácticas de PIAT, y generar documentación que sirva como recurso para estudiantes y profesores interesados en utilizar JavaCC en proyectos relacionados con el procesamiento de información.

3.6. Conclusiones

En esta sección se ha presentado una introducción al desarrollo de la propuesta. En las siguientes secciones se desarrollarán los aspectos relativos a la implementación, los procedimientos seguidos y el entorno utilizado para los desarrollos y las pruebas.

3.7. Análisis de ficheros de *log*. Práctica 2

3.7.1. Introducción

La practica 2 de PIAT tiene como objetivo aplicar los conocimientos adquiridos sobre expresiones regulares y su uso en la programación Java para analizar y extraer información de logs de un sistema de correo electrónico.

En concreto, el objetivo de la practica es utilizar expresiones regulares para extraer información estadística y generar informes a partir de los archivos de log del sistema de correo electrónico. La arquitectura del sistema y el formato de los logs se detallan en un documento anexo.

La práctica se divide en tres partes principales: La obtención de estadísticos generales, cuya función es contabilizar el número de servidores procesados, los archivos procesados, registros procesados y registros con errores de formato; El análisis por tipo de servidor y día, en el que para cada tipo de servidor, se debe calcular y mostrar los siguientes estadísticos por día:

- msgIn: Número de mensajes entrantes.
- msgOut: Número de mensajes salientes.
- msgINFECTED: Número de mensajes infectados con virus.
- msgSPAM: Número de mensajes SPAM no bloqueados.
- code 4.3.2: Número de intentos de entrega con código de estado 4.3.2 (sobrecarga).
- code 5.1.1: Número de intentos de entrega de mensajes entrantes con código de estado 5.1.1 (dirección de correo electrónico de destino incorrecta).

Y por ultimo, identificar aquellas cuentas de correo internas desde las que se han enviado más de 500 mensajes. Para cada cuenta, se debe mostrar el nombre de usuario y el número de mensajes enviados.

Un ejemplo del resultado a obtener se encuentra en el **anexo**.

3.7.2. Expresiones Regulares. RegEx

3.7.3. Desarrollo de la práctica

A continuación se presenta la estructura de árbol que sigue el analizador sintáctico para determinar el tipo de traza que esta analizando y las acciones correspondientes a realizar en caso de encontrar un tipo de traza u otra.

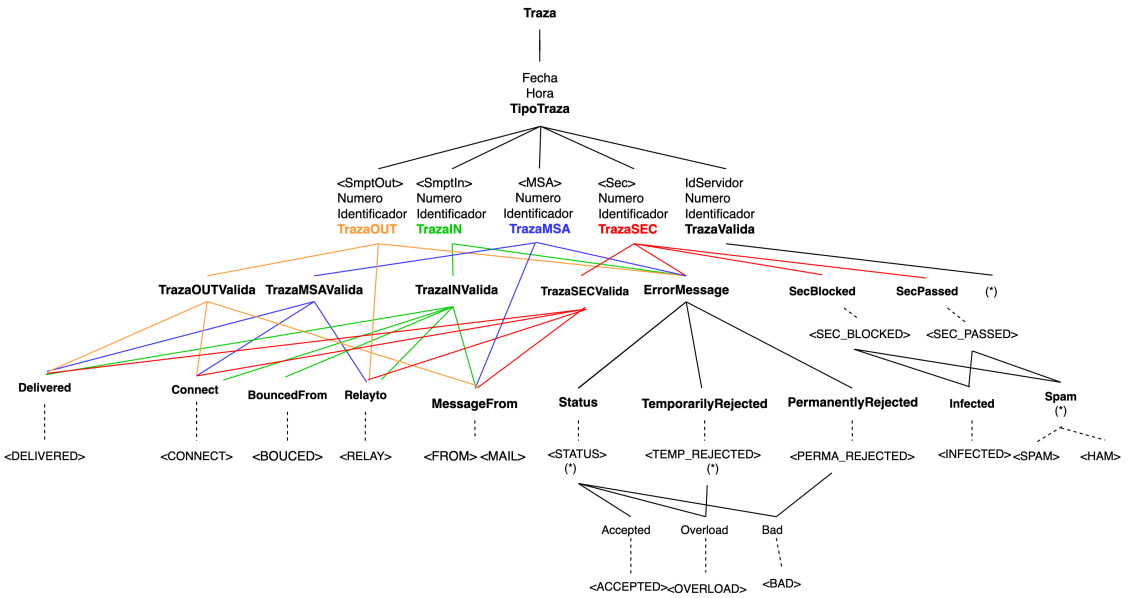

Figura 3: Estructura de árbol. Análisis de trazas log.

En esta figura se observa toda la estructura que se ha definido a la hora de reconocer trazas. Se observa como se contemplan cinco tipos de trazas, las `msa`, `smpt-in`, `smpt-out`, `security` y otros tipos de servidores. Para cada tipo de traza se definen las distintas posibilidades que puede haber dentro de cada tipo de traza, de tal forma que se describe la estructura de forma detallada. En aquellas "ramas" en las que queramos ejecutar alguna acción, como puede ser añadir los valores a un estadístico, se define una acción léxica dentro del no-terminal.

```
void Overload() : {}
{
   <OVERLOAD>
   { Registar("code 4.3.2",t, n, f); }
}
```

3.7.4. Validación de resultados

3.8. Análisis de archivos XML. Práctica 3

3.8.1. Introducción

La práctica 3 de PIAT, centrada en el procesamiento de documentos XML, representa un hito significativo en el marco de este proyecto. En esta etapa, se

aborda la implementación de un analizador de documentos XML utilizando la tecnología SAX. El objetivo principal es familiarizar a los estudiantes con SAX y fortalecer sus habilidades en el diseño de algoritmos eficientes para la extracción y transformación de información a partir de fuentes de contenidos estructurados. La práctica 3 de PIAT se centra en familiarizarse con la tecnología SAX y desarrollar un analizador de documentos XML basado en esta tecnología. Además, el objetivo secundario es diseñar algoritmos eficientes que permitan la extracción y transformación de información a partir de fuentes de contenidos estructurados.

3.8.2. SAX

La tecnología SAX (Simple API for XML) es una API que se encarga de procesar documentos XML. Está basada en eventos, lo que quiere decir que el analizador XML notifica al programa cuando encuentra un evento especifico en el documento. El programa puede tomar medidas en función del evento. Los eventos que puede notificar SAX son:

- Inicio de documento: Se produce cuando el analizador comienza a procesar el documento.
- Fin de documento: Se produce cuando el analizador finaliza el procesamiento del documento.
- Inicio de elemento: Se produce cuando el analizador encuentra el inicio de un elemento XML.
- Fin de elemento: Se produce cuando el analizador encuentra el final de un elemento XML.
- Caracteres: Se produce cuando el analizador encuentra caracteres no pertenecientes a un elemento XML.
- Error: Se produce cuando el analizador encuentra un error en el documento XML.

SAX es una buena opción para procesar documentos XML grandes o complejos. Esto se debe a que el analizador XML no necesita cargar todo el documento en memoria a la vez. En cambio, el analizador puede procesar el documento de forma secuencial, lo que puede ahorrar memoria.

SAX también es una buena opción para procesar documentos XML que se actualizan con frecuencia. Esto se debe a que el analizador XML puede procesar solo los cambios en el documento, lo que puede ahorrar tiempo.

Algunos ejemplos de cómo se puede utilizar SAX para procesar documentos XML incluyen:

- Leer un documento XML y extraer los datos que contiene.
- Validar un documento XML para asegurarse de que cumple con las reglas de un esquema XML.
- Transformar un documento XML a un formato diferente.

3.8.3. JavaCC vs SAX

JavaCC y SAX son dos API diferentes para procesar documentos XML. JavaCC es una herramienta de generación de analizadores que permite crear analizadores XML personalizados. SAX es una API estándar que proporciona un conjunto de métodos para procesar documentos XML. Hay varias razones por las que JavaCC puede ser una mejor opción que SAX para procesar documentos XML:

- Control: JavaCC ofrece un mayor control sobre el proceso de análisis que SAX. Esto permite a los desarrolladores crear analizadores que se adapten a sus necesidades específicas.
- Eficiencia: JavaCC puede generar analizadores que sean más eficientes que los analizadores SAX estándar. Esto se debe a que JavaCC puede aprovechar las características del lenguaje Java para optimizar el proceso de análisis.
- Flexibilidad: JavaCC permite crear analizadores XML que sean más flexibles que los analizadores SAX estándar. Esto se debe a que JavaCC permite a los desarrolladores definir sus propias reglas de análisis.

En concreto, JavaCC ofrece las siguientes ventajas sobre SAX:

- Puede generar analizadores personalizados que se adapten a las necesidades específicas del documento XML. Esto permite a los desarrolladores crear analizadores que sean más eficientes, precisos y fáciles de mantener.
- Puede generar analizadores que sean más eficientes que los analizadores SAX estándar. Esto se debe a que JavaCC puede aprovechar las características del lenguaje Java para optimizar el proceso de análisis.
- Puede generar analizadores que sean más flexibles que los analizadores SAX estándar. Esto se debe a que JavaCC permite a los desarrolladores definir sus propias reglas de análisis.

Sin embargo, JavaCC también tiene algunas desventajas con respecto a SAX:

- Es más complejo de aprender y usar que SAX. Esto se debe a que JavaCC requiere conocimientos de programación en Java.
- Puede ser más lento que SAX para documentos XML pequeños. Esto se debe a que JavaCC debe generar un analizador personalizado para cada documento XML.

En general, JavaCC es una buena opción para procesar documentos XML cuando se necesita un mayor control, eficiencia o flexibilidad. Sin embargo, SAX es una buena opción para procesar documentos XML pequeños o cuando se necesita una solución rápida y fácil de usar.

3.8.4. Descripción de la práctica

A continuación se presenta el enunciado de la práctica propuesta del año 2022: Se desea desarrollar una aplicación que extraiga cierta información de un fichero XML obtenido a partir de los datos publicados en el Portal de Datos Abiertos del Ayuntamiento de Madrid (http://datos.madrid.es).

La información (organismos, eventos, actividades, . . .) publicada a través del Portal de Datos Abiertos presenta las siguientes características:

- Los elementos de información, en adelante recursos (concept), se identifican mediante una URI.
- Cada recurso se encuentra asociado a una categoría (elemento code de concept).
- Los recursos se encuentran agrupados en conjuntos de datos (elemento dataset) accesibles en formato JSON a través de un URI indicada en el atributo id del elemento dataset.
- En un dataset puede haber información sobre recursos asociados a varias categorías.
- Los recursos de una categoría pueden estar accesibles a través de diferentes datasets.
- Para la categorización de los recursos se utiliza un sistema de clasificación

jerárquico basando en características temáticas.

Esta información se encuentra descrita en un Catálogo de Datos en formato XML (catalogo.xml) válido con respecto al esquema catalogo.xsd. La aplicación por desarrollar proporcionará una herramienta de búsqueda que posibilite la recuperación de recursos asociados a un código de categoría generando un documento XML con los resultados. La Figura 1 muestra un ejemplo de presentación de parte de la estructura jerárquica de los concepts del catálogo.

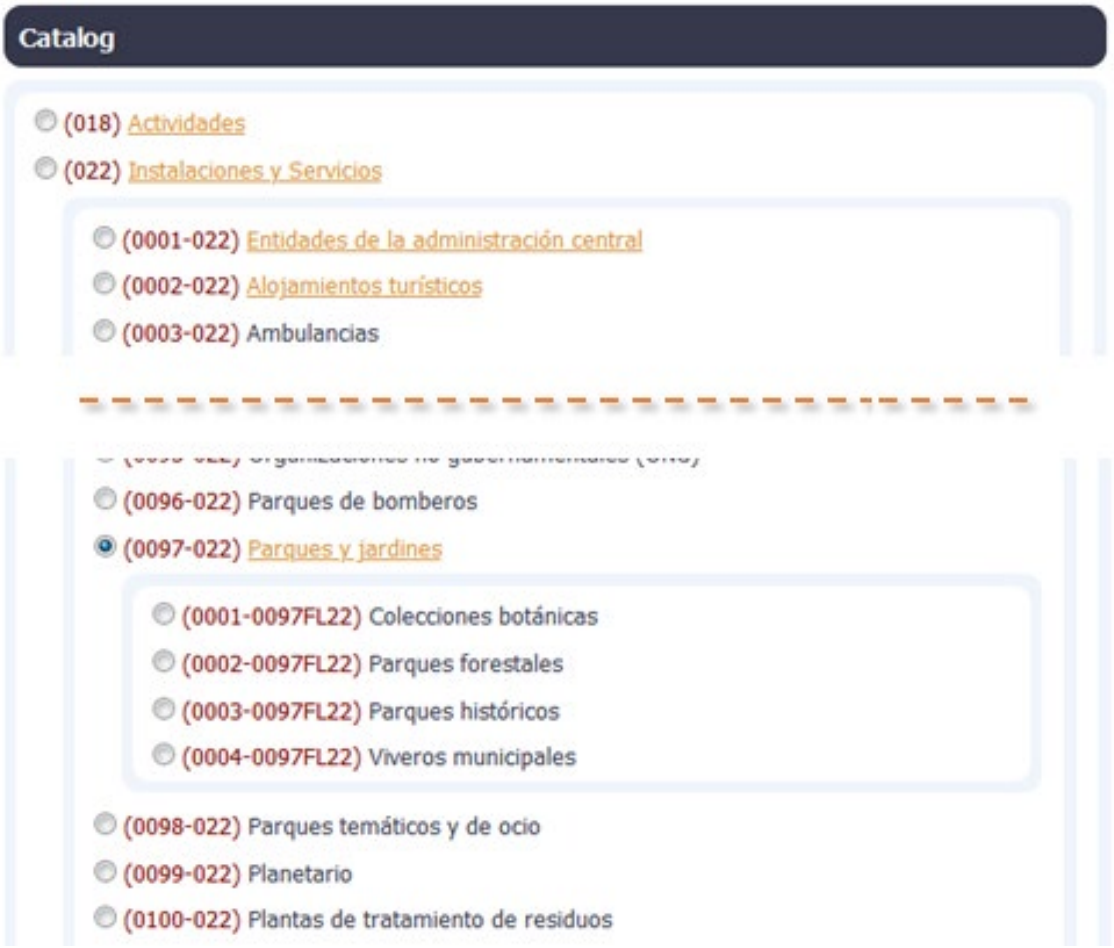

Figura 4: Representación de la estructura jerárquica de los concepts del catálogo de datos.

La aplicación por desarrollar recibirá como argumento el criterio de búsqueda, esto es, el código de la categoría de la que se desea información, y proporcionará información sobre los concepts y datasets pertinentes, aplicando para ello los siguientes criterios:

- Se considerarán pertinentes el concept cuyo código (elemento code) coincida con el criterio de búsqueda y todos los concepts descendientes del mismo.
- Se considerarán pertinentes los dataset que contengan información asociada a alguno de los concept pertinentes (contengan un elemento concept con el atributo id igual al atributo id del elemento concept pertinente).

La Figura 2 muestra un ejemplo de búsqueda del concept con código 0097-022 y los resultados que se obtendrían.

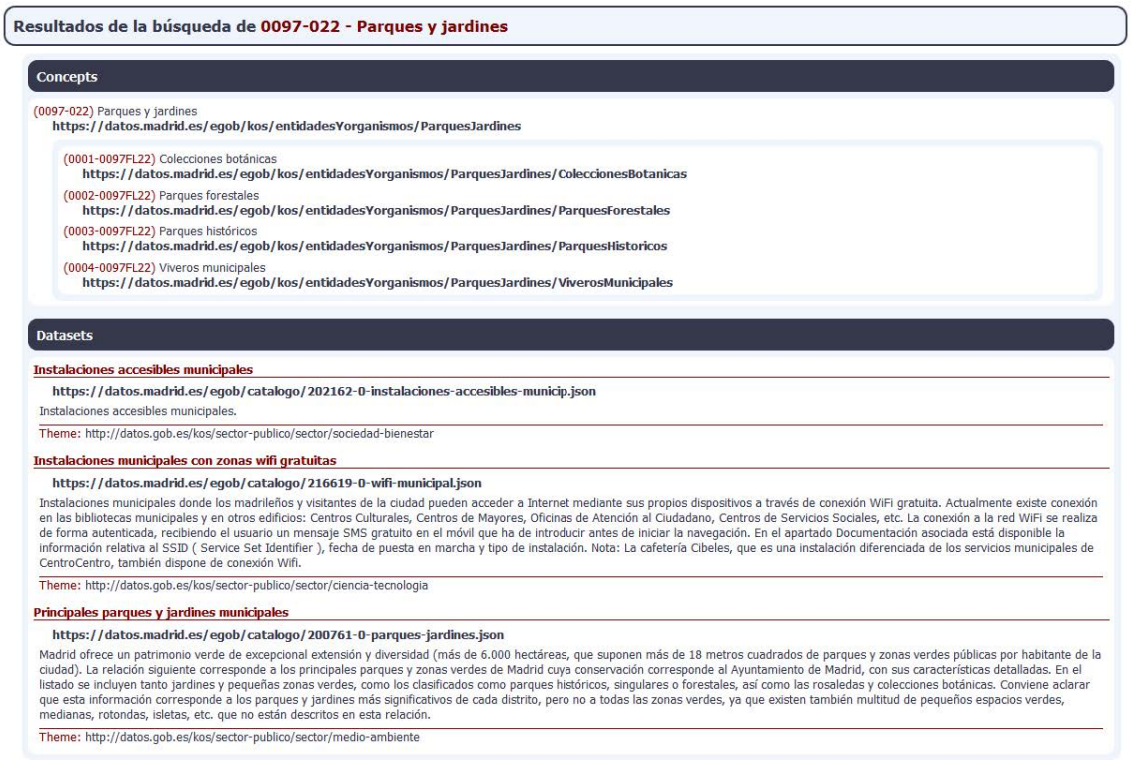

Figura 5: Concepts y datasets pertinentes para el código 0097-022.

3.8.5. Desarrollo de la práctica

La implementación de la Práctica 3 se basa en la creación de un analizador de documentos XML denominado XMLParser, desarrollado utilizando JavaCC, una herramienta que permite generar analizadores léxicos y sintácticos en el lenguaje Java. Este analizador se ha diseñado para procesar documentos XML que contienen datos publicados en el Portal de Datos Abiertos del Ayuntamiento de Madrid (http://datos.madrid.es). Los documentos XML que se procesan a través de esta herramienta presentan ciertas características clave, como la identificación de recursos mediante URI, la asociación a categorías, la agrupación en conjuntos de datos y una clasificación jerárquica basada en características temáticas. La tarea principal del analizador es extraer información específica relacionada con códigos de categorías y, posteriormente, generar un nuevo documento XML que contiene los resultados deseados.

3.8.6. Herramienta

La principal herramienta desarrollada en esta práctica es el analizador XMLParser. Este analizador es capaz de llevar a cabo varias tareas esenciales:

- Validación de argumentos de entrada para garantizar que los parámetros cumplan con las especificaciones requeridas.
- Extracción de información relevante de los documentos XML, siguiendo las reglas y estructuras definidas en la gramática.
- Generación de documentos de resultados que cumplen con un esquema específico.

El analizador XMLParser representa una contribución significativa en términos de

procesamiento y manipulación de documentos XML.

3.8.7. Conclusiones

La Práctica 3 ha brindado a los estudiantes una experiencia valiosa en el manejo de tecnologías de análisis de documentos XML. A través del uso de JavaCC y la tecnología SAX, se ha proporcionado una base sólida para procesar información estructurada de manera eficiente. Los estudiantes han tenido la oportunidad de aplicar conceptos relacionados con la validación de argumentos, el análisis de elementos XML y la generación de documentos de resultados. Uno de los aspectos más destacados de esta práctica es la capacidad de filtrar y seleccionar información relevante basada en códigos de categorías. Esta habilidad es fundamental en situaciones donde la extracción selectiva de datos es esencial.

3.8.8. Preguntas Frecuentes

¿Qué sentido tiene utilizar la clase ManejadorXML, si el parser ya busca lo que quiere? Tiene sentido cronológico, ya que primero el parser realiza el análisis del documento y devuelve los datos, si se llama a una función que devuelva los datos es posible que se llamen a estas funciones antes de que se llame a la función de análisis

3.8.9. Notas

> hacer clase concept con atributos code y label, hacer lista concepts de concept. hacer lo mismo con dataset. recoger los datos al igual que hacíamos en NXCalculator Concepts, son objetos distintos, se tienen que reconocer de manera diferente. ->Nuevo objeto conceptInDataset No hace falta un estado léxico, simplemente crear un objeto nuevo ya que son cosas diferentes, ->IncludedConcepts String id ->idConcepts XMLParser devuelve un objeto que tenga implementado la interfaz ParserCatalogo

3.8.10. Código Fuente

El código fuente de los archivos elaborados en esta practica se encuentra a continuación para su estudio y aprendizaje. Cabe destacar que se proporcionan dicho código para que el lector sea capaz de probar las funcionalidades con el objetivo de aprender mas acerca de las aplicaciones de JavaCC.

No se promueve el plagio y la realización de las prácticas, este simplemente es una de las muchas formas en las que se podría realizar la práctica.

XMLParser.jj

3.9. Análisis de archivos JSON. Práctica 4

3.9.1. Introducción

La Práctica 4 se centra en la exploración y aplicación de la tecnología GSON Streaming para el análisis y procesamiento eficiente de documentos JSON. El

objetivo principal de esta etapa es familiarizar a los estudiantes con la API GSON Streaming y desarrollar un analizador de documentos JSON basado en esta tecnología. Como objetivo secundario, se busca capacitar a los estudiantes en el diseño de algoritmos eficientes para la extracción y transformación de información proveniente de fuentes de contenidos estructurados.

3.9.2. GSON Streaming

La API de transmisión de GSON es una API de Java que permite leer y escribir JSON de forma secuencial. Esto la hace útil en situaciones donde no es posible o deseable cargar el modelo de objeto completo en memoria, como cuando se trabaja con grandes cantidades de datos o cuando los datos se reciben de forma continua.

La API de transmisión de GSON se basa en dos clases principales: JsonReader y JsonWriter. JsonReader se utiliza para leer JSON de forma secuencial, mientras que JsonWriter se utiliza para escribir JSON de forma secuencial.

Las características principales de la API son su eficiencia en términos de memoria y su flexibilidad. Por una parte, la API de transmisión de GSON no necesita cargar el modelo de objeto completo en memoria, lo que la hace más eficiente en términos de memoria que la API de GSON tradicional. Además, GSON Streaming permite leer y escribir datos JSON de forma secuencial, lo que la hace muy flexible.

La API es ideal para trabajar con grandes cantidades de datos, ya que puede leer y escribir los datos sin necesidad de cargarlos todos en memoria. También es ideal para recibir datos de forma continua, ya que puede leer los datos a medida que se reciben.

3.9.3. JavaCC vs GSON Streaming

3.9.4. Descripción de la práctica

La práctica busca ampliar la funcionalidad de la herramienta de búsqueda del Portal de Datos Abiertos del Ayuntamiento de Madrid, desarrollada en la Práctica 3. Se pretende dotar a esta herramienta de la capacidad para extraer información sobre los recursos asociados a una categoría específica (concept) del portal. Esto se logrará accediendo a los conjuntos de datos (dataset) en formato JSON y procesándolos con un analizador GSON Streaming.

El código desarrollado en la Práctica 3 servirá como punto de partida, y se completará para integrar un analizador GSON Streaming. Este analizador procesará la información de cada conjunto de datos pertinente y generará un documento XML válido conforme al esquema de documento ResultadosBusquedaP4.xsd. El nuevo elemento introducido en este esquema es resources, que contendrá información sobre los recursos asociados a las categorías pertinentes.

La generación del documento XML (indicado por ARG2) será similar al proceso de la Práctica 3, pero ahora se incorporará el elemento resources. Se utilizará la clase JSONDatasetParser para implementar el analizador GSON Streaming, y se analizarán los archivos .json indicados en el atributo id de cada dataset. Se añadirán como máximo cinco recursos (resource) a partir de cada dataset analizado.

3.9.5. Desarrollo de la práctica

3.9.6. Herramienta

La herramienta principal desarrollada en esta práctica es la extensión de la herramienta de búsqueda del Portal de Datos Abiertos, ahora mejorada con la capacidad de analizar y procesar datos JSON mediante la API GSON Streaming. La clase JSONDatasetParser representa la implementación de este analizador GSON Streaming. La herramienta final permite la extracción selectiva de información de archivos JSON, generando un documento XML conforme al esquema ResultadosBusquedaP4.xsd que incluye el nuevo elemento resources.

3.9.7. Conclusiones

La Práctica 4 ha proporcionado a los estudiantes una oportunidad única para aplicar sus conocimientos adquiridos en la Práctica 3 y explorar en profundidad la API GSON Streaming. La capacidad de extender la funcionalidad de la herramienta existente para manejar datos JSON y generar documentos XML enriquecidos demuestra una comprensión avanzada de las tecnologías de procesamiento de datos estructurados.

El uso de GSON Streaming versión 2.9.0 ha permitido a los estudiantes implementar un analizador eficiente para manejar grandes conjuntos de datos JSON, cumpliendo así con los requisitos de la práctica.

3.9.8. Preguntas Frecuentes

Pregunta 1: ¿Cuál es el propósito principal de la Práctica 4 en términos de procesamiento de información JSON? Respuesta: El objetivo principal es extender la herramienta de búsqueda para extraer información sobre recursos asociados a categorías específicas en el Portal de Datos Abiertos, utilizando la tecnología GSON Streaming.

Pregunta 2: ¿Qué elemento se introduce en el esquema Resultados BusquedaP4.xsd ? Respuesta: Se introduce el elemento resources, que contendrá información sobre los recursos asociados a las categorías pertinentes.

Pregunta 3: ¿Cuál es la versión de la API GSON Streaming utilizada en esta práctica? Respuesta: Se utiliza la versión 2.9.0 de la API GSON Streaming, disponible en este enlace.

3.9.9. Notas

3.9.10. Código Fuente

El código fuente de los archivos elaborados en esta practica se encuentra a continuación para su estudio. Cabe destacar que se proporcionan dicho código para que el lector sea capaz de probar las funcionalidades con el objetivo de aprender mas acerca de las aplicaciones de JavaCC.

No se promueve el plagio y la realización de las prácticas, este simplemente es una de las muchas formas en las que se podría realizar la práctica.

JSONParser.jj

3.10. Evolutivo de las prácticas

3.10.1. Introducción

Como se ha podido observar de las prácticas de las secciones anteriores, pese a ser interesantes y cumplir con los objetivos establecidos de aprendizaje, son poco interesantes y se podría realizar unos ejercicios mas potentes y versátiles que exprimiesen al máximo las funcionalidades de las herramientas que hemos ido repasando.

Es por ello que en esta sección se plantea unos ejercicios a modo de evolutivo, que amplían los requerimientos y complejidad del análisis. Además se observará que, gracias al incremento en la complejidad de los ejercicios, aumenta en proporción la sencillez con la que se resuelven dichos problemas empleando JavaCC.

De esta forma, se podrá observar lo ventajoso de crear un analizador a la medida de usuario, y como JavaCC es una herramienta versátil y muy potente.

3.10.2. Evolutivo 1. XML

Este evolutivo se presenta a partir de la Practica 3. En dicha práctica se analizaban documentos xml, y se proponía analizar un documento xml con cierta recursividad

XPath, abreviatura de XML Path Language, es un lenguaje de consulta para documentos XML. Se utiliza para seleccionar partes de un documento XML, como elementos, atributos, texto y datos binarios.

XPath se basa en una sintaxis similar a la de las expresiones regulares. Las expresiones XPath se utilizan para construir caminos a través de un documento XML.

Las principales características de XPath son las siguientes:

Selección de elementos: XPath se puede utilizar para seleccionar elementos individuales o conjuntos de elementos en un documento XML. Selección de atributos: XPath se puede utilizar para seleccionar atributos individuales o conjuntos de atributos de un elemento. Selección de texto: XPath se puede utilizar para seleccionar texto de un elemento o atributo. Selección de datos binarios: XPath se puede utilizar para seleccionar datos binarios de un elemento o atributo.

XPath se utiliza en una variedad de aplicaciones, incluyendo el procesamiento de XML, el desarrollo web, y la integración de datos. Esto permite que se utilize para procesar documentos XML, extraer datos, validar documentos y transformar documentos. Además, se utiliza en aplicaciones web para acceder a datos XML, como datos de formularios o datos de un servicio web. Por último, también se puede emplear para integrar datos de diferentes fuentes, como datos XML y datos de bases de datos.

3.10.3. JavaCC vs XPath

JavaCC y XPath son dos herramientas que se pueden utilizar para procesar documentos XML. Sin embargo, tienen algunas diferencias clave.

JavaCC es una herramienta de generación de analizadores sintácticos. Se utiliza para generar analizadores sintácticos que pueden analizar documentos XML.

Como ya hemos visto en secciones anteriores, las principales ventajas de JavaCC son su eficiencia en términos de memoria y su gran flexibilidad a los usuarios. Por otra parte, la principal desventaja de JavaCC es su dificultad de uso,

XPath es un lenguaje de consulta para documentos XML. Se utiliza para seleccionar partes de un documento XML.

Las principales ventajas de XPath son su sencillez, ya que es una herramienta relativamente sencilla de aprender a utilizar, y su generalidad, pues se puede utilizar para procesar cualquier documento XML, independientemente de su lenguaje.

Las principales desventajas de XPath son su eficiencia, debido a que es menos eficiente en términos de memoria que JavaCC, y sus limitaciones, como la imposibilidad de procesar datos binarios.

3.10.4. Descripción de la práctica

La implementación de la Práctica 5 se basa en extender el código desarrollado en la Práctica 4, integrando la funcionalidad de XPath para extraer información específica del documento XML resultante. El uso del Modelo de Objetos de Documento (DOM) facilita la navegación y manipulación de la estructura jerárquica del documento XML.

Se requerirá la introducción de expresiones XPath para identificar y seleccionar los elementos deseados en el documento XML. La información obtenida mediante XPath se almacenará en un nuevo documento en formato JSON. Este proceso de transformación garantiza que la información relevante se extraiga eficientemente y se represente en un formato JSON para su posterior análisis o intercambio.

3.10.5. Desarrollo de la práctica

3.10.6. Herramienta

La herramienta principal desarrollada en esta práctica es una extensión del código existente en la Práctica 4, ahora mejorado con la capacidad de utilizar expresiones XPath para extraer información específica del documento XML resultante. La información extraída se guarda en un nuevo documento en formato JSON, ofreciendo una representación alternativa y flexible de los datos.

3.10.7. Conclusiones

La Práctica 5 proporciona a los estudiantes una valiosa experiencia en la utilización de XPath para la extracción selectiva de información de documentos XML. La capacidad de integrar esta tecnología con el código existente de la Práctica 4 demuestra la versatilidad en el manejo de datos estructurados.

El proceso de transformación a JSON destaca la flexibilidad de las tecnologías utilizadas, permitiendo la representación de datos en diferentes formatos según las necesidades del proyecto.

3.10.8. Preguntas Frecuentes

Pregunta 1: ¿Cuál es el propósito principal de la Práctica 5 en términos de transformación de datos? Respuesta: El objetivo principal es la extracción de información específica del documento XML mediante expresiones XPath y la posterior representación de esta información en un nuevo documento en formato JSON.

Pregunta 2: ¿Cómo se realiza la transformación de datos de XML a JSON en esta práctica? Respuesta: La transformación se realiza mediante la utilización de expresiones XPath para identificar y seleccionar información específica en el documento XML generado en la Práctica 4. La información seleccionada se almacena en un nuevo documento en formato JSON.

Pregunta 3: ¿Cómo se integra la funcionalidad XPath en el código existente de la Práctica 4? Respuesta: Se realizarán modificaciones en el código de la Práctica 4 para incorporar expresiones XPath que seleccionen la información deseada. La información extraída se utilizará para generar un nuevo documento en formato JSON

3.10.9. Notas

Capítulo 4

Resultados

Capítulo 5

Conclusiones

Apéndice A

Preguntas Frecuentes

A.1. ¿Que es una producción BNF o ENBF?

En informática, una producción BNF o EBNF es una regla que define cómo se puede generar una secuencia de símbolos en un lenguaje formal. Las producciones BNF y EBNF se utilizan para definir la sintaxis de los lenguajes de programación, los sistemas de comandos y los protocolos de comunicación.

A.1.1. BNF

La notación de Backus-Naur (BNF) es un metalenguaje utilizado para expresar gramáticas libres de contexto. Una gramática libre de contexto es un tipo de gramática formal que se caracteriza por la ausencia de reglas de recursión izquierda.

Una producción BNF se compone de dos partes: el lado izquierdo y el lado derecho. El lado izquierdo de una producción es el símbolo que se está definiendo. El lado derecho de una producción es una expresión que especifica cómo se puede generar el símbolo del lado izquierdo.

Las expresiones en el lado derecho de una producción BNF pueden ser:

Símbolos simples: un símbolo simple es un símbolo que no se puede descomponer en otros símbolos. Por ejemplo, el símbolo + es un símbolo simple.

Secuencias de símbolos: una secuencia de símbolos es una lista de símbolos separados por espacios. Por ejemplo, la secuencia de símbolos a b c es una secuencia de tres símbolos.

Alternativas: una alternativa es una lista de opciones separadas por barras verticales (|). Por ejemplo, la alternativa a | b | c significa que el símbolo del lado izquierdo puede ser a, b o c.

Recursión derecha: una recursividad derecha es una producción que se refiere a sí misma en el lado derecho. Por ejemplo, la producción S → S + E | E significa que el símbolo S puede ser S más E o E.

A.1.2. EBNF

La notación EBNF (Extended Backus-Naur Form) es una extensión de la notación BNF. La notación EBNF incluye algunas características adicionales que hacen que sea más fácil de usar para definir la sintaxis de los lenguajes de programación.

Las principales características adicionales de la notación EBNF son:

Recursiva izquierda: la notación EBNF permite la recursividad izquierda, lo que hace que sea más fácil de definir la sintaxis de los lenguajes de programación que

utilizan recursividad izquierda, como los lenguajes de programación orientados a objetos.

Referencia de producciones: la notación EBNF permite referenciar producciones existentes en el lado derecho de una producción. Esto hace que sea más fácil de definir la sintaxis de los lenguajes de programación que utilizan estructuras de datos complejas, como los árboles.

Operadores de conjuntos: la notación EBNF incluye operadores de conjuntos que permiten especificar conjuntos de símbolos. Esto hace que sea más fácil de definir la sintaxis de los lenguajes de programación que utilizan conjuntos de símbolos, como los lenguajes de programación de patrones.

Ejemplos

Aquí hay algunos ejemplos de producciones BNF y EBNF:

BNF

$$S \rightarrow E$$
$$E \rightarrow T + E \mid T$$
$$T \rightarrow id \mid num$$

Estas producciones definen la sintaxis de una expresión aritmética simple. El símbolo S representa una expresión, el símbolo E representa un término y el símbolo T representa un factor.

EBNF

$$S ::= E$$
$$E ::= T'' + ''E \mid T$$
$$T ::= id \mid num$$

Estas producciones son equivalentes a las producciones BNF anteriores.

Otro ejemplo

$$S \rightarrow ''(''E'')''|id$$
$$E \rightarrow E'' + ''E|E'' - ''E|E'' * ''E|E''/''E|id$$

Estas producciones definen la sintaxis de una expresión aritmética más compleja. El símbolo S representa una expresión, el símbolo E representa un término y el símbolo id representa un identificador.

Conclusiones

Las producciones BNF y EBNF son herramientas importantes para la definición de la sintaxis de los lenguajes formales. Las producciones BNF son más simples de usar, pero las producciones EBNF ofrecen algunas características adicionales que pueden ser útiles para definir la sintaxis de lenguajes de programación complejos.

A.2. ¿Qué es un no-terminal?

En informática, un no terminal es un símbolo que representa un conjunto de cadenas de caracteres. Los no terminales se utilizan en gramáticas formales para definir la sintaxis de los lenguajes formales.

Un no terminal se representa generalmente con una letra mayúscula. Por ejemplo, el símbolo E podría representar el conjunto de todas las expresiones aritméticas.

Las reglas de una gramática formal definen cómo se pueden generar las cadenas de caracteres que representan los no terminales. Estas reglas se llaman producciones.

Por ejemplo, la siguiente producción define cómo se puede generar el no terminal E:

$$E \rightarrow T + E \mid T$$

Esta producción significa que una expresión aritmética (E) puede ser una suma de dos términos (T + E) o un solo término (T).

Los no terminales son importantes porque permiten definir la sintaxis de los lenguajes formales de una manera jerárquica. Esto facilita la comprensión de la sintaxis de un lenguaje y la construcción de analizadores sintácticos que puedan verificar si una cadena de caracteres es válida para un lenguaje determinado.

Ejemplos de no terminales

Aquí hay algunos ejemplos de no terminales:

En la gramática de expresiones aritméticas, el símbolo E representa el conjunto de todas las expresiones aritméticas. En la gramática de oraciones en español, el símbolo SN representa el conjunto de todos los sintagmas nominales. En la gramática de programas en Python, el símbolo stmt representa el conjunto de todas las declaraciones.

Conclusiones

Los no terminales son una herramienta fundamental para la definición de la sintaxis de los lenguajes formales. Permiten definir la sintaxis de un lenguaje de una manera jerárquica y facilita la comprensión y la construcción de analizadores sintácticos.

A.3. ¿Cómo empiezo a desarrollar en JavaCC?

En el *Apéndice B* se encuentra una guía y manual para empezar a desarrollar sus proyectos con esta herramienta. En ella se encuentran contenidos de instalación, configuración y primeros pasos.

A.4. ¿Dónde puedo encontrar información adicional?

La mejor fuente de información sobre JavaCC es Internet. Hay muchos sitios web que ofrecen información sobre JavaCC, incluidos tutoriales, artículos y ejemplos.

Uno de los sitios web más útiles es la página web oficial de JavaCC. Esta página web contiene documentación completa sobre JavaCC, incluidas las especificaciones de la sintaxis de JavaCC, ejemplos de uso y una lista de preguntas frecuentes.

En la bibliografía de este documento se incluyen algunos libros y páginas web de referencia que pueden ser útiles para aprender más sobre JavaCC.

Estos recursos incluyen:

El libro “JavaCC: The Java Compiler Compiler” de Sanjiva Weerawarana El libro “JavaCC: A Tutorial” de Scott Ambler La página web de JavaCC Tutorial

Documentación oficial de JavaCC

La documentación oficial de JavaCC es una fuente importante de información sobre la herramienta. Esta documentación incluye las especificaciones de la sintaxis de JavaCC, ejemplos de uso y una lista de preguntas frecuentes.

Apéndice B

Herramientas utilizadas

B.1. Manual de instalación y configuración

En este apartado vamos a desarrollar los manuales necesarios para la instalación de la herramienta de JavaCC, el IDE de desarrollo que queramos utilizar, y el plugin de JavaCC para el IDE correspondiente.

B.1.1. Instalación de JavaCC

Se puede utilizar JavaCC directamente desde la línea de comandos, o puede optar por utilizar un IDE, que permite desarrollar proyectos con rapidez [16] e integra varias herramientas para desarrollar proyectos de forma más eficiente y productiva.

Descarga e Instalación de JavaCC desde línea de comandos

Descarga

Descargue la última versión estable (al menos el código fuente y los binarios). Actualmente la versión estable más reciente es la 7.0.13, por lo que la descarga del binario se accedería a través de:

https://repo1.maven.org/maven2/net/java/dev/javacc/javacc/7.0.13/javacc.jar

y descargar el archivo `javacc-7.0.13.jar`

La descarga del código fuente se realiza a través del repositorio oficial de Java CC:

https://github.com/javacc/javacc/releases

Instalación

Una vez que haya descargado los archivos, navegue hasta el directorio de descarga y descomprima el archivo fuente, creando así el llamado directorio de instalación de JavaCC:

```
unzip javacc-7.0.13.zip
```

o

```
tar xvf javacc-7.0.13.tar.gz
```

A continuación, mueva el archivo binario bajo el directorio de descarga en un nuevo directorio bajo el directorio de instalación y cámbiele el nombre a `javacc-7.0.13.jar target/javacc.jar`

Posteriormente, añada el directorio del directorio de instalación de JavaCC a su archivo . Los scripts/ejecutables de invocación de JavaCC, JJTree y JJDoc residen en este directorio `scripts/PATH` En los sistemas basados en UNIX, es posible que

los scripts no se puedan ejecutar inmediatamente. Esto se puede resolver usando el comando en el directorio `javacc-7.0.13/`:

```
chmod +x scripts/javacc
```

En caso de tener un sistema operativo macOS, la instalación de javacc se facilita en gran medida, ya que se puede realizar directamente mediante brew:

```
brew install javacc
```

Descarga e Instalación de JavaCC desde un IDE

Para poder usar JavaCC en un IDE se necesita como mínimo que el IDE tenga soporte para Java, y soporte para Maven con Java. IDEs como IntelliJ o Eclipse son compatibles con JavaCC mediante la instalación de un complemento para su desarrollo.

Descarga de IntelliJ: https://www.jetbrains.com/idea/

Plugin IntelliJ JavaCC: https://plugins.jetbrains.com/plugin/11431-javacc/

Descarga de Eclipse: https://www.eclipse.org/ide/

Plugin Eclipse JavaCC: https://marketplace.eclipse.org/content/javacc-eclipse-plug

Para Maven, hay que añadir la siguiente dependencia al archivo `pom.xml`:

```
<dependency>
    <groupId>net.java.dev.javacc</groupId>
    <artifactId>javacc</artifactId>
    <version>7.0.13</version>
</dependency>
```

En el caso de utilizar un IDE, deberá descargarse el binario, al igual que si desease desarrollar JavaCC desde la línea de comandos. Para saber el procedimiento a seguir, visite *Descarga e Instalación de JavaCC desde línea de comandos*.

Configuración de JavaCC en Eclipse

Una vez instalado JavaCC, para poder desarrollar proyectos utilizando la JavaCC en Eclipse hay que seguir los siguientes pasos:

1. Crear un proyecto nuevo o elegir uno existente.
2. Abrir las propiedades del proyecto (Alt+Enter) (cmd+Enter en macOS)
3. Ir a *JavaCC > Global Options*, y en *Set the default JavaCC jar file*, poner la ruta al binario descargado anteriormente (`javacc.jar`)

Si mueve el binario descargado a la ruta por defecto, no es necesario realizar el paso 3. Es recomendable poner el binario en la ruta por defecto ya que, de lo contrario, para cada proyecto que quiera desarrollar va a tener que realizar este procedimiento. La ruta por defecto se indica debajo de *"Set the default JavaCC jar file"* del paso 3, (default: plugin's jar).

Si la configuración se ha realizado correctamente, a la hora de guardar los archivos JavaCC, se generarán la compilación de las clases Java correspondientes.

```
>java -classpath C:\javacc\javacc-javacc-7.0.12\target\javacc.jar javacc XMLParser.jj (@ 22/01/2024 17:59:20)
Java Compiler Compiler Version 7.0.12 (Parser Generator)
(type "javacc" with no arguments for help)
Reading from file XMLParser.jj . . .
File "Provider.java" is being rebuilt.
File "StringProvider.java" is being rebuilt.
File "StreamProvider.java" is being rebuilt.
File "TokenMgrException.java" is being rebuilt.
File "ParseException.java" is being rebuilt.
File "Token.java" is being rebuilt.
File "SimpleCharStream.java" is being rebuilt.
Parser generated successfully.
```

Figura 6: Compilación de Archivos JavaCC en Eclipse

B.2. Símbolos de Expresiones regulares en JavaCC

+ : El símbolo + se usa para indicar que **un elemento puede aparecer una o más veces**. Por ejemplo, si tienes `A+`, significa que se espera que haya al menos una instancia de `A`, pero puede haber más.

* : El símbolo * se usa para indicar que **un elemento puede aparecer cero o más veces**. Por ejemplo, si tienes `B*`, significa que `B` es opcional y puede aparecer cero o más veces.

? : El símbolo? se usa para indicar que **un elemento puede aparecer cero o una vez**. Por ejemplo, si tienes `C?`, significa que `C` es opcional y puede aparecer cero o una vez.

~ : El símbolo ~ se utiliza para **excluir ciertos caracteres** o elementos. Por ejemplo, `~A` significa cualquier carácter excepto `A`. En una expresión regular, ~ se usa para negar un conjunto de caracteres. Por ejemplo, `~[0-9]` significa cualquier carácter que no sea un dígito del 0 al 9.

Apéndice C

Código Fuente

En este apéndice se presenta el código fuente de los archivos referenciados a lo largo del documento para su estudio. Cabe destacar que se proporciona dicho código con el objetivo de que el lector sea capaz de probar las funcionalidades y aprender mas acerca de las aplicaciones de JavaCC.

No se promueve el plagio y la realización de las prácticas, este simplemente es una de las muchas formas en las que se podrían realizar la prácticas.

La realización de las prácticas y de todos los ejemplos mostrados en este documento se encuentran disponibles en:

https://github.com/m-amaris/tfg/tree/main/eclipse-workspace

C.1. Mathexp.jj

```
// Define la gramática para analizar una simple expresión matemática.
void Expression() : {}
{
  Term() ( "+" Term() | "-" Term() )*
}

void Term() : {}
{
  Factor() ( "*" Factor() | "/" Factor() )*
}

void Factor() : {}
{
  <NUMBER>
  | "(" Expression() ")"
}
```

C.2. NL_Xlator.jj

```
options {
  STATIC = false;
}
PARSER_BEGIN(NL_Xlator)
/**
 * New line translator.
 */
public class NL_Xlator
{
  /** Main entry point. */
  public static void main(String args []) throws ParseException
  {
    NL_Xlator parser = new NL_Xlator(System.in);
    parser.ExpressionList();
  }
}

PARSER_END(NL_Xlator)

SKIP :
{
  " "
| "\t"
| "\n"
| "\r"
}

TOKEN :
{
  < ID : [ "a"-"z", "A"-"Z", "_" ] ([ "a"-"z", "A"-"Z", "_", "0"-"9" ])* >
|
  < NUM : ([ "0"-"9" ])+ >
}

/** Top level production. */
void ExpressionList() :
{
    double e;
}
{
    {
      System.out.println("Please type in an expression followed by a \";\" or ^D to
    quit:");
      System.out.println("");
    }
  (
    e = Expression() ";"
      {
        System.out.println("Resultado =" + e);
```

```
        System.out.println("");
        System.out.println("Please type in another expression followed by a \";\"
   or ^D to quit:");
        System.out.println("");
      }
  )*
  < EOF >
}

/** An Expression. */
double Expression() :
{
    double e = 0, t = 0;
}
{
  t = Term()
    {
      e = t;
    }
  (
    "+" t = Term()
      {
        e = e + t;
      }
  |
    "-" t = Term()
      {
        e = e - t;
      }
  )*
  {
    return e;
  }
}

/** A Term. */
double Term() :
{
    double f = 0, t = 0;
}
{
  f = Factor()
    {
      t = f;
    }
  (
    "*" f = Factor()
      {
        t = t * f;
      }
  |
    "/" f = Factor()
```

```
      {
        t = t / f;
      }
  )*
  {
    return t;
  }
}

/** A Factor. */
double Factor() :
{
    Token t;
    double e;
}
{
  t = < NUM >
    {
      return Double.parseDouble(t.image);
    }
|
  "(" e = Expression() ")"
    {
      return e;
    }
}
```

C.3. Catalogo.xml (Simplificado)

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog xmlns="http://www.piat.upm.es/catalogo" xmlns:xsi="http://www.w3.org/2001/
   XMLSchema-instance" xsi:schemaLocation="http://www.piat.upm.es/catalogo catalogo
   .xsd ">
  <concepts>
    <concept id="https://datos.madrid.es/egob/kos/actividades">
      <code>018</code>
      <label><![CDATA[Actividades]]></label>
      <concepts>
        <concept id="https://datos.madrid.es/egob/kos/actividades/1
   ciudad21distritos">
          <code>0001-018</code>
          <label><![CDATA[CiudaDistrito]]></label>
        </concept>
        <concept id="https://datos.madrid.es/egob/kos/actividades/
   ActividadesCalleArteUrbano">
          <code>0002-018</code>
          <label><![CDATA[Actividades calle, arte urbano]]></label>
        </concept>
        <concept id="https://datos.madrid.es/egob/kos/actividades/
   ActividadesDeportivas">
          <code>0003-018</code>
          <label><![CDATA[Actividades deportivas]]></label>
          <concepts>
            <concept id="https://datos.madrid.es/egob/kos/actividades/
   ActividadesDeportivas/Ajedrez">
              <code>0001-0003FL18</code>
              <label><![CDATA[Ajedrez]]></label>
            </concept>
          </concepts>
        </concept>
        <concept id="https://datos.madrid.es/egob/kos/actividades/
   ActividadesEscolares">
          <code>0004-018</code>
          <label><![CDATA[Actividades para escolares]]></label>
        </concept>
        <concept id="https://datos.madrid.es/egob/kos/actividades/
   CineActividadesAudiovisuales">
          <code>0008-018</code>
          <label><![CDATA[Cine actividades audiovisuales]]></label>
          <concepts>
            <concept id="https://datos.madrid.es/egob/kos/actividades/
   CineActividadesAudiovisuales/CineDocumental">
              <code>0001-0008FL18</code>
              <label><![CDATA[Cine documental]]></label>
            </concept>
            <concept id="https://datos.madrid.es/egob/kos/actividades/
   CineActividadesAudiovisuales/CineExperimental">
              <code>0002-0008FL18</code>
```

```
            <label><![CDATA[Cine experimental]]></label>
          </concept>
        </concepts>
      </concept>
    </concepts>
  </concept>
</concepts>
<datasets>
  <dataset id="https://datos.madrid.es/egob/catalogo/212827-0-administracion-
 justicia.json">
    <title><![CDATA[Sedes. Administración de Justicia]]></title>
    <description><![CDATA[Relación de juzgados, tribunales y otras sedes de la
 Administración de Justicia en la ciudad de Madrid. Se aportan datos de dirección
 , transportes, horarios de atención y coordenadas para su georreferenciación.
 Importante: La información proporcionada puede no ser exhaustiva, es la
 disponible actualmente en madrid.es.]]></description>
    <theme><![CDATA[http://datos.gob.es/kos/sector-publico/sector/sector-publico
 ]]></theme>
    <publisher><![CDATA[http://datos.gob.es/recurso/sector-publico/org/Organismo/
 L01280796]]></publisher>
    <concepts>
      <concept id="https://datos.madrid.es/egob/kos/entidadesYorganismos/
 ArchivosCentrosDocumentacion"/>
      <concept id="https://datos.madrid.es/egob/kos/entidadesYorganismos/
 EntidadesAdministracionJusticia/AudienciasTribunales"/>
      <concept id="https://datos.madrid.es/egob/kos/entidadesYorganismos/
 EntidadesAdministracionJusticia/Juzgados"/>
      <concept id="https://datos.madrid.es/egob/kos/entidadesYorganismos/
 EntidadesAdministracionJusticia"/>
      <concept id="https://datos.madrid.es/egob/kos/entidadesYorganismos/
 MonumentosEdificiosArtisticos"/>
    </concepts>
  </dataset>
  <dataset id="https://datos.madrid.es/egob/catalogo/212816-0-investigacion.json"
 >
    <title><![CDATA[Sedes. Institutos de Investigación]]></title>
    <description><![CDATA[Relación de Centros e Institutos de investigación en la
  ciudad de Madrid, tanto de ámbito público como privado. Importante: La
 información proporcionada puede no ser exhaustiva, es la disponible actualmente
 en madrid.es.]]></description>
    <theme><![CDATA[http://datos.gob.es/kos/sector-publico/sector/ciencia-
 tecnologia]]></theme>
    <publisher><![CDATA[http://datos.gob.es/recurso/sector-publico/org/Organismo/
 L01280796]]></publisher>
    <concepts>
      <concept id="https://datos.madrid.es/egob/kos/entidadesYorganismos/
 CentrosInstitutosInvestigacion"/>
      <concept id="https://datos.madrid.es/egob/kos/entidadesYorganismos/
 Planetarios"/>
    </concepts>
  </dataset>
</datasets>
```

```
</catalog>
```

C.4. Catalogo.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.piat.upm.es/catalogo"
    xmlns:tns="http://www.piat.upm.es/catalogo"
    elementFormDefault="qualified">

<xsd:element name="catalog">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns:concepts" />
      <xsd:element ref="tns:datasets" />
    </xsd:sequence>
  </xsd:complexType>

  <xsd:key name="keyConcept">
    <xsd:selector xpath="./tns:concepts/tns:concept/tns:concepts/tns:concept/
   tns:concepts/tns:concept | ./tns:concepts/tns:concept/tns:concepts/tns:concept |
    ./tns:concepts/tns:concept "/>
    <xsd:field xpath="@id"/>
  </xsd:key>

  <xsd:keyref name="refDSConcept" refer="tns:keyConcept">
    <xsd:selector xpath="./tns:datasets/tns:dataset/tns:concepts/tns:concept" />
    <xsd:field xpath="@id" />
  </xsd:keyref>

  <xsd:key name="keyConceptCod">
    <xsd:selector xpath=".//tns:code"/>
    <xsd:field xpath="."/>
  </xsd:key>
</xsd:element>

<xsd:element name="concepts">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="concept" type="tns:tConcept" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

<!--
  <xsd:key name="keyConceptCod">
    <xsd:selector xpath=" ./tns:concept/tns:concepts/tns:concept/tns:concepts/
   tns:concept/tns:concepts/tns:concept | ./tns:concept/tns:concepts/tns:concept/
   tns:concepts/tns:concept  | ./tns:concept/tns:concepts/tns:concept | ./
   tns:concept   "/>
    <xsd:field xpath="./tns:code"/>
  </xsd:key>
  -->
```

```
</xsd:element>

<xsd:complexType name="tConcept">
  <xsd:sequence>
    <xsd:element name="code" type="xsd:string"/>
    <xsd:element name="label" type="xsd:string"/>
    <xsd:element ref="tns:concepts" minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="id" type="xsd:anyURI"/>
</xsd:complexType>


<xsd:element name="datasets">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="dataset" type="tns:tDataset" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:key name="keyDataset">
    <xsd:selector xpath="./tns:dataset"/>
    <xsd:field xpath="@id"/>
  </xsd:key>

</xsd:element>

<xsd:complexType name="tDataset">
  <xsd:all>
    <xsd:element name="title" type="xsd:string" />
    <xsd:element name="description" type="xsd:string" minOccurs="0" />
    <xsd:element name="keyword" type="xsd:string" minOccurs="0"/>
    <xsd:element name="theme" type="xsd:string" minOccurs="0"/>
    <xsd:element name="publisher" type="xsd:string" minOccurs="0"/>
    <xsd:element name="concepts" minOccurs="0">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="concept" maxOccurs="unbounded">
            <xsd:complexType>
              <xsd:attribute name="id" type="xsd:anyURI"/>
            </xsd:complexType>
          </xsd:element>
        </xsd:sequence>
      </xsd:complexType>

      <xsd:key name="keyDatasetConcept">
        <xsd:selector xpath="./tns:concept"/>
        <xsd:field xpath="@id"/>
      </xsd:key>
    </xsd:element>
  </xsd:all>
  <xsd:attribute name="id" type="xsd:anyURI"/>
```

```
</xsd:complexType>

</xsd:schema>
```

C.5. XMLParser.jj

```
options
{
  STATIC = false;
  JAVA_TEMPLATE_TYPE = "modern";
}

PARSER_BEGIN(XMLParser)
package piat.opendatasearch;
import java.util.List;
import java.util.ArrayList;
public class XMLParser{
  private List < Concept > conceptsList;
  private List < Dataset > datasetsList;
  private String conceptLabel;
  private String nombreCategoria;
}
PARSER_END(XMLParser)

SKIP :
{
  " "
| "\t"
| "\r"
| "\n"
}

TOKEN :
{
  < OPEN_CONCEPTS : "<concepts>" >
| < OPEN_DATASETS : "<datasets>" >
| < CATALOG : "<catalog" >
| < XMLNS : "xmlns=" >
| < XMNLNS_XSI : "xmlns:xsi=" >
| < XSI_SCHEMA_LOCATION : "xsi:schemaLocation=" >
| < HEADER : "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" >
| < OPEN_CONCEPT : "<concept" >
| < OPEN_DATASET : "<dataset" >
| < CLOSE_CONCEPTS : "</concepts>" >
| < CLOSE_DATASETS : "</datasets>" >
| < CLOSE_CATALOG : "</catalog>" >
| < CLOSE_CONCEPT : "</concept>" >
| < CLOSE_DATASET : "</dataset>" >
| < OPEN_CODE : "<code" >
| < CLOSE_CODE : "/code>" >
| < OPEN_LABEL : "<label>" >
| < OPEN_TITLE : "<title>" >
| < OPEN_DESCRIPTION : "<description>" >
| < OPEN_KEYWORD : "<keyword>" >
| < OPEN_THEME : "<theme>" >
```

```
| < OPEN_PUBLISHER : "<publisher>" >
| < CLOSE_LABEL : "</label>" >
| < CLOSE_TITLE : "</title>" >
| < CLOSE_DESCRIPTION : "</description>" >
| < CLOSE_KEYWORD : "</keyword>" >
| < CLOSE_THEME : "</theme>" >
| < CLOSE_PUBLISHER : "</publisher>" >
| < ID : "id=" >
| < VALUE_CODE : ">" (~[ "\"", "\n", "\r", "<", ">", "=" ])+ "<" >
| < VALUE_LABEL : "<![CDATA[" (~[ "\"", "\n", "\r", "<", ">", "=" ])+ "]]>" >
| < STRING : "\"" (~[ "\"", "\n", "\r" ])+ "\"" >
| < END_ELEMENT : ">" >
}

SKIP :
{
  < ~[ ] >
}

ManejadorXML processFile(String code) :
{
  conceptsList = new ArrayList < Concept > ();
  datasetsList = new ArrayList < Dataset > ();
  nombreCategoria = code;
}
{
  < HEADER >
  < CATALOG > < XMLNS > < STRING > < XMNLNS_XSI > < STRING > < XSI_SCHEMA_LOCATION
   > < STRING > < END_ELEMENT >
  (< OPEN_CONCEPTS > (concept())+ < CLOSE_CONCEPTS >)+
  (< OPEN_DATASETS > (dataset())+ < CLOSE_DATASETS >)+
  < CLOSE_CATALOG >
  {
    return new ManejadorXML(datasetsList,conceptsList);
  }
}

Concept concept() :
{
  Token idValue;
  String codeValue, labelValue;
  Concept childConcept;
  List < Concept > concepts = new ArrayList < Concept > ();
}
{
  < OPEN_CONCEPT > < ID > idValue = < STRING > < END_ELEMENT >
  < OPEN_CODE > codeValue = getCode() < CLOSE_CODE >
  < OPEN_LABEL > labelValue = getValue() < CLOSE_LABEL >
  (
    < OPEN_CONCEPTS >
    (
      childConcept = concept()
```

```
      {
        concepts.add(childConcept);
      }
    )+
    < CLOSE_CONCEPTS >
  )?
  < CLOSE_CONCEPT >
  {
    if(codeValue.equals(nombreCategoria)) {
      conceptLabel = idValue.image;
      conceptsList.add(new Concept(idValue.image.replace("\"", ""), codeValue,
   labelValue, concepts));
  }
  return new Concept(idValue.image.replace("\"", ""), codeValue, labelValue,
   concepts);
  }
}


void dataset() :
{
  Token idDataset, idConcept;
  String title;
  String description = "";
  String keyword = "";
  String theme = "";
  String publisher = "";
  List < IdConcept > idConcepts = new ArrayList < IdConcept > ();
}
{
  < OPEN_DATASET > < ID > idDataset = < STRING > < END_ELEMENT >
  < OPEN_TITLE > title = getValue() < CLOSE_TITLE >
  (< OPEN_DESCRIPTION > description = getValue() < CLOSE_DESCRIPTION >)?
  (< OPEN_KEYWORD >  keyword = getValue() < CLOSE_KEYWORD >)?
  (< OPEN_THEME > theme = getValue() < CLOSE_THEME >)?
  (< OPEN_PUBLISHER > publisher = getValue() < CLOSE_PUBLISHER >)?
  (
    < OPEN_CONCEPTS >
    (
       < OPEN_CONCEPT > < ID > idConcept = < STRING > < END_ELEMENT >
      {
        idConcepts.add(new IdConcept(idConcept.image));
      }
    )+
    < CLOSE_CONCEPTS >
  )?
  < CLOSE_DATASET >
  {
    for(IdConcept ic : idConcepts) {
      if(ic.getId().equals(conceptLabel)) {
      datasetsList.add(new Dataset(idDataset.image.replace("\"", ""), title,
   description, keyword,theme,publisher,idConcepts));
```

```
      }
    }
    //return new Dataset(idDataset.image, title, description, keyword,theme,
   publisher,idConcepts);
  }

}

String getCode() :
{
  Token t = new Token();
}
{
  (t = < VALUE_CODE >)?
  {
    return t.image.substring(1, t.image.length() - 1);
  }
}

String getValue() :
{
  Token t = new Token();
}
{
  (t = < VALUE_LABEL >)?
  {
    return t.image;
  }
}
```

C.6. JSONParser.jj

```
options
{
  STATIC = false;
  JAVA_TEMPLATE_TYPE = "modern";
  UNICODE_INPUT=true; // Permite reconocer caracteres Unicode sin que todo explote
   :) Especificar el encoding (UTF-8)
}

PARSER_BEGIN(JSONParser)
package piat.opendatasearch;
import java.util.List;
public class JSONParser {}
PARSER_END(JSONParser)

/*****************************  ANALIZADOR LÉXICO  *****************************/

/** Cuando el token @graph es detectado, empieza a analizar el graph.
  Obvia todo excepto el token @graph.
  ---------------------------------------------------------------
  When a @graph token is detected, it starts to analyze the graph.
  Skips everything but the @graph token. **/
TOKEN :
{
  < GRAPHS : "\"@graph\"" > : GRAPH
}

/** Cuando el token @id es detectado, empieza a analizar el recurso.
  Cuando el token "]" es detectado, sale del graph.
  Obvia todo excepto los tokens @id, "[" y "]".
  ---------------------------------------------------------------
  When the @id token is detected, it starts to analyze the resource.
  When the "]" token is detected, it exits the graph.
  Skips everything but the @id, "[", "]" tokens. **/
< GRAPH >
TOKEN :
{
  < IDRESOURCE : "\"@id\"" > : GETVALUE
| < ARRAY_START : "[" >
}

/** Analiza los tokens que nos interesan dentro del recurso.
  Cuando el token "organization-name" es detectado, le dice al analizador léxico
   que el recurso va a terminar (RESOURCE_END).
  Cuando el token "area" es detectado, cambia el estado léxico para detectar el
   token @id dentro de "area" (INAREA).
  Obvia todo excepto los tokens que nos interesan.
  ---------------------------------------------------------------
  Analyzes every token of the resource that we are interested on.
```

```
  When the "organization-name" token is detected, it tells the lexical analyzer
   that the resource is about to end (RESOURCE_END).
  When the "area" token is detected, it changes the lexical state to detect the @id
    token inside the "area" (INAREA).
  Skips everything but the tokens we are interested on. **/
< RESOURCE >
TOKEN :
{
  < IDCONCEPT : "\"@type\"" > : GETVALUE
| < TITLE : "\"title\"" > : GETVALUE
| < DESCRIPTION : "\"description\"" > : GETVALUE
| < TIME_START : "\"dtstart\"" > : GETVALUE
| < TIME_END : "\"dtend\"" > : GETVALUE
| < LINK : "\"link\"" > : GETVALUE
| < EVENT_LOCATION : "\"event-location\"" > : GETVALUE
| < AREA : "\"area\"" > : INAREA
| < LATITUDE : "\"latitude\"" > : GETVALUE
| < LONGITUDE : "\"longitude\"" > : GETVALUE
| < ORGANIZATION_NAME : "\"organization-name\"" > : GETVALUE
| < ACCESIBILITY : "\"accesibility\"" > : GETVALUE
| < ARRAY_END : "]" > : DEFAULT
}

/** Analiza cuando es el final de un rescurso, y comienza el siguiente.
  Cuando el token es detectado, sale del recurso, y vuelve al graph.
  Obvia todo excepto el token ORGANIZATION_STRING
  ---------------------------------------------------------------
  Analyze the end of the resource and the beginning of a new one.
  When the token is detected, it exits the resource, and gets back to the graph.
  Skips everything but the ORGANIZATION_STRING token. **/
< RESOURCE>
SKIP:
{
< END_LAST_OBJECT: "},\n  {"  > : GRAPH
}

/** Estado léxico que recoge el valor del los tokens dentro del recurso.
  Cuando cualquier token es detectado, vuelve al estado de análisis del recurso.
  Obvia todo excepto los tokens que nos interesan.
  ---------------------------------------------------------------
  Lexical state that gets the value of tokens in the resource.
  When any token is detected, it jumps back to the analyzing state of the resource
   .
  Skips everything but the tokens we are interested on. **/
< GETVALUE >
TOKEN :
{
  < STRING : "\"" (~[ "\"", "\n", "\r" ])* "\"" > : RESOURCE
| < INTEGER : ("-")? ([ "0"-"9" ])+ > : RESOURCE
| < DOUBLE : ("-")? ([ "0"-"9" ])+("." ([ "0"-"9" ])+) >: RESOURCE
| < TRUE : "true" > : RESOURCE
| < FALSE : "false" > : RESOURCE
```

```
| < NULL : "null" > : RESOURCE
}

/** Cuando el token @id es detectado dentro del area, recoge su valor.
  Obvia todo excepto el token @id.
  ----------------------------------------------------------------
  When the @id token is detected inside the area, it gets the value.
  Skips everything but the @id token. **/
< INAREA >
TOKEN :
{
  < IDAREA : "\"@id\"" > : GETVALUE
}

/** Todo carácter que no es un token, es obviado.
  ----------------------------------------------------------------
  Any character that is not a token, it is skipped. **/
<* >
SKIP :
{
  < ~[ ] >
}

/*****************************  ANALIZADOR SINTÁCTICO  *****************************/

void processFile(List < String > conceptsList, List < Resource> resourceList) :
{
  //List < Resource > resourceList = new ArrayList < Resource > ();
  Resource r;
  int maxObjects = 5;
}
{
  (
    < GRAPHS > < ARRAY_START >
    (
      r = resource()
      {
      if (conceptsList.contains(r.getConceptId())) {
        resourceList.add(r);
        }

        if (resourceList.size() >= maxObjects)
        {
              token_source.SwitchTo(DEFAULT); // No es muy seguro, es posible que
   el TokenManager vaya muy por delante del analizador.
                              // Pero dado que no queremos analizar nada mas, se
   puede utilizar sin problemas.
            break;
        }
      }
    )*
    (< ARRAY_END >)?
```

```
  )

}

Resource resource() :
{
  Resource r = new Resource();
  Token t;
}
{
  (< IDRESOURCE > t = < STRING >){r.setResourceId(t.image.replace("\"", ""));}
  (< IDCONCEPT >  t = < STRING >)?{r.setConceptId(t.image.replace("\"", ""));}
  (< TITLE >  t = < STRING >){r.setTitle(t.image.replace("\"", ""));}
  (< DESCRIPTION >  t = < STRING >){r.setDescription(t.image.replace("\"", ""));}
  (< TIME_START > t = < STRING > {r.setStartDate(t.image.replace("\"", "")); })
  (< TIME_END > t = < STRING > {r.setEndDate(t.image.replace("\"", "")); })
  (< LINK >  t = < STRING >){r.setLink(t.image.replace("\"", ""));}
  (< EVENT_LOCATION >  t = < STRING >){r.setEventLocation(t.image.replace("\"", "")
   );}
  (< AREA>(< IDAREA > t = < STRING >){r.setArea(t.image.replace("\"", ""));})?
  (< LATITUDE >  t = < DOUBLE >)?{if(!t.image.toString().equals("\"\""))r.
   setLatitude(Double.parseDouble(t.image));}
  (< LONGITUDE >  t = < DOUBLE >)?{if(!t.image.toString().equals("\"\""))r.
   setLongitude(Double.parseDouble(t.image));}
  (< ORGANIZATION_NAME > t = < STRING >)?{r.setOrganizationName(t.image.replace("\"
   ", ""));}
  (< ACCESIBILITY >  t = < STRING >)?{r.setAccesibility(t.image.replace("\"", ""))
   ;}

  {
    return r;
  }
}
```

Bibliografía

[1] S. Viswanadha y S. Sankar. «JavaCC. The most popular parser generator for use with Java applications.» (1996), dirección: https://javacc.github.io/javacc/.

[2] *Análisis léxico (analizador) en diseño de compilador con ejemplo*. dirección: https://www.guru99.com/es/compiler-design-lexical-analysis.html.

[3] *Analizador Léxico con JavaCC*. dirección: https://www.youtube.com/watch?v=MlBmgcwFLEc.

[4] *Analizador léxico*. dirección: https://es.wikipedia.org/wiki/Analizador_l%C3%A9xico#Token.

[5] J. Ágila, *Funciones del Analizador Sintáctico*. dirección: https://kataix.umag.cl/~jaguila/Compilers/T09_Funciones_Sintacticas.pdf.

[6] A. Sierra, S. Gálvez, R. Tema y S. Análisis, «Traductores, Compiladores e Intérpretes,» dirección: http://www.lcc.uma.es/~galvez/ftp/tci/tictema3.pdf.

[7] *Analizador Sintáctico con JavaCC*. dirección: https://www.youtube.com/watch?v=GdWn6v0t4uc&list=PLKuOYUvjPrsOLt8gdgk9G4Q3LjVZoUWpW&index=2&t=146s.

[8] *Apache Ant JavaCC Task*. dirección: https://ant.apache.org/manual/Tasks/javacc.html.

[9] G. Succi y R. Wong, «The application of JavaCC to develop a C/C++ preprocessor,» *ACM Sigapp Applied Computing Review*, vol. 7, págs. 11-18, sep. de 1999. DOI: 10.1145/333630.333633.

[10] V. Kodaganallur, «Incorporating language processing into Java applications: A JavaCC tutorial,» *Software, IEEE*, vol. 21, págs. 70-77, ago. de 2004. DOI: 10.1109/MS.2004.16.

[11] *Apache Projects Directory*. dirección: https://projects.apache.org/.

[12] *Apache HTTP Server: ¿Qué es, cómo funciona y para qué sirve? | Blog IBX Agency*. dirección: https://www.ibxagency.com/blog/apache-http-server-que-es-como-funciona-y-para-que-sirve/.

[13] *Expression Language - The Java EE 6 Tutorial*. dirección: https://docs.oracle.com/javaee/6/tutorial/doc/gjddd.html.

[14] *JavaParser*. dirección: https://javaparser.org/.

[15] *What's better, ANTLR or JavaCC? - Stack Overflow*. dirección: https://stackoverflow.com/questions/382211/whats-better-antlr-or-javacc.

[16] *¿Qué es y para qué sirve un IDE?* Dirección: https://www.redhat.com/es/topics/middleware/what-is-ide.

[17] A. V. Aho, *Compiladores : principios, técnicas y herramientas*. Pearson Educación, 2008, ISBN: 9789702611332.

[18] *introducción a JavaCC*. dirección: https://studylib.es/doc/6172839/introducci%C3%B3n-a-javacc.

[19] *Mi primer proyecto utilizando JavaCC - Erick Navarro*. dirección: https://www.ericknavarro.io/2020/02/10/23-Mi-primer-proyecto-utilizando-JavaCC/.

[20] *GitHub - RobertFischer/json-parser: JavaCC-built JSON Parser*. dirección: https://github.com/RobertFischer/json-parser.

[21] *Implementing the XQuery grammar*. dirección: https://xqengine.sourceforge.net/extreme_2002.html.

[22] *Introducing the JavaCC 21 parser generator for Java*. dirección: https://blogs.oracle.com/javamagazine/post/introducing-the-javacc-21-parser-generator-for-java.

[23] *JavaCC Parser Generator Integration Tutorial for the NetBeans Platform*. dirección: https://netbeans.apache.org/tutorial/main/tutorials/nbm-javacc-parser/.