Memoria

Contenido

[ApiCreatorXML V0.1 1](#_Toc14716364)

[Clase ApiCreatorXML V0.1 2](#_Toc14716365)

[Clase Node V0.1 2](#_Toc14716366)

[Clase Paso V0.1 2](#_Toc14716367)

[Clase Writer V0.1 3](#_Toc14716368)

[ApiCreatorXML V0.2 6](#_Toc14716369)

[Clase ApiCreatorXML V0.1 6](#_Toc14716370)

[Clase Node V0.2 6](#_Toc14716371)

[Clase Paso V0.2 6](#_Toc14716372)

[Clase Writer V0.2 7](#_Toc14716373)

[ApiCreatorXML V0.3 13](#_Toc14716374)

[Clase ApiCreatorXML V0.1 13](#_Toc14716375)

[Clase Node V0.3 13](#_Toc14716376)

[Clase Paso V0.2 13](#_Toc14716377)

[Clase Writer V0.3 14](#_Toc14716378)

[ApiCreatorXML V0.4 20](#_Toc14716379)

[Clase ApiCreatorXML V0.1 20](#_Toc14716380)

[Clase Node V0.3 20](#_Toc14716381)

[Clase Paso V0.2 20](#_Toc14716382)

[Clase Writer V0.4 21](#_Toc14716383)

[ApiCreatorXML V0.5 27](#_Toc14716384)

[Clase ApiCreatorXML V0.1 27](#_Toc14716385)

[Clase Node V0.3 27](#_Toc14716386)

[Clase Paso V0.2 28](#_Toc14716387)

[Clase Writer V0.5 28](#_Toc14716388)

[Clase Annotator V0.1 36](#_Toc14716389)

# ApiCreatorXML V0.1

## Clase ApiCreatorXML V0.1

Se utiliza para comprobar el funcionamiento de la api. Crea el escritor y escribe la parte del traductor. Añade los nodos y escribe la parte del árbol. Añade una serie de pasos y escribe la parte de contenido. Por último, crea el XML.

## Clase Node V0.1

Representa los nodos de la parte de árbol del XML

**Atributos:**

**private Integer id:** id del nodo, se crea automáticamente según vas añadiendo nodos.

**private String element:** elemento del nodo.

**private Boolean terminal:** true si es terminal false si no lo es.

**private Integer nivel:** niveldel árbol en el que se encuentra.

**Constructor:**

**public Node(Integer id, String element, Boolean terminal, Integer nivel):**

Recoge los datos pertenecientes a todos los atributos y los asigna.

## Clase Paso V0.1

**Atributos:**

**private Integer id:** representa el id del paso, el cual se asigna automáticamente cuando se añaden los pasos.

**private String tipo:** describe el tipo de paso(“DespDes” para desplazamientos en descendentes, “derivacion” para derivaciones, “desplazamiento” para desplazamientos en ascendentes y “reduccion” para reducciones ).

**private String leído:** parte de la cadena leída.

**private String pendiente:** parte dela cadena pendiente.

**private String elemento:** elemento que se está procesando.

**private String valor:** valor del elemento que se está procesando.

**private String regla:** regla correspondiente al paso (dicha regla solo se añade si es el primer elemento de la regla )

**private Integer widthRule**: ancho de la regla (pendiente de eliminar)

**private Integer relNodo:** nodo que produce este paso. (paso padre).

**private ArrayList<Integer> changedNodes:** nodos que cambian después de este paso.

**private ArrayList<String> changes:** valor de los pasos que van a cambiar después de este paso.

**Constructor:**

**public Paso(Integer id, String tipo, String leido, String pendiente, String elemento, String valor,Integer relNodo, String regla, Integer widthRule)**

Recoge los valores de los atributos e inicializa las listas **changedNodes** y **changes.**

## Clase Writer V0.1

**Atributos:**

**private HashMap<String, ArrayList<String>> grammarWithActions:** corresponde a la gramática , es un mapa que contiene las producciones por cada antecedente.

**private String path:** ruta de la gramatica

**private String pathResult:** ruta donde guardar el xml

**private String entryChain:** cadena de entrada

**private Element espec:** corresponde al elemento a partir del que se va a escribir las partes del traductor , la cadena , el árbol y el contenido.

**private Document doc:** representa el documento XML

**private Integer ruleCount:** contador de reglas.

**private ArrayList<String> antecedentes:** lista con los antecedentes de la gramática.

**private Integer numNodos:** contador de nodos

**private ArrayList<Node> nodes:** lista de nodos

**private ArrayList<Paso> steps:** lista de pasos

**private HashMap<String, String> ruleId:** mapa con los ids de cada regla

**private Integer pasoCount:** contador de pasos

**Constructor:**

**public Writer( String path, String pathResult, String entryChain):**

Recoge la ruta de la gramática, la ruta de destino y la cadena de entrada.

Inicializa los mapas de la gramática y de los ids por regla y las listas de antecedentes, pasos y nodos. Lee la gramática y la guarda en los mapas. Inicializa el documento XML.

**Métodos:**

**public Node addNode(String element, Boolean terminal, Integer nivel):**

Crea un nodo, lo añade a la lista de nodos y aumenta el contador de nodos. Devuelve el nodo creado.

**public Paso addPaso(String tipo,String leido, String pendiente,String element, String valor, String regla,Integer relNodo)**

Crea un paso, lo añade a la lista de pasos y aumenta el contador de pasos. Devuelve el paso creado.

**public Boolean writeXML():**

Crea el archivo XML.

**public String removeActions(String production):**

Dada una producción elimina las acciones y los símbolos recursivos.

Se separa la producción y por cada símbolo si no es una acción, si pertenece a los antecedentes se añade al resultado , si no pertenece se añade al resultado la parte del símbolo que termina cuando empiezan los dígitos.

**public String removeOnlyActions(String production):**

Dada una producción elimina las acciones.

Se separa la producción y por cada símbolo si no es una acción se añade al resultado.

**private Integer getNumberIndex(String symbol):**

Devuelve el índice donde empiezan los dígitos en un símbolo recursivo.

Por cada letra se suma uno hasta que se detecta un digito.

**private void readFile():**

Lee y guarda la gramática desde un archivo de texto.

Por cada línea la separa en antecedente y producciones y se separa las producciones. Se añade el antecedente a la lista si no está y se guarda en el mapa de la gramática con acciones una lista nueva con el antecedente como clave. Se añaden cada producción a la lista anteriormente mencionada.

**public void writeTraductor():**

Escribe la parte de <traductor> y <cadena> del XML.

Se crea la etiqueta “traductor” y se añade a la etiqueta espec. Se crea la etiqueta “tipo” y se añade a traductor, después se añade el texto “Descendente” a tipo. Por cada antecedente , por cada producción se añade la regla. Cuando se han añadido todas las reglas se crea la etiqueta “cadena” y se añade a espec, por ultimo se añade el texto de la cadena de entrada a cadena.

**private void addRule(String antecedent ,String production,Element traductor):**

Dado un antecedente , una producción y la etiqueta a la que añadir la regla , añade una regla al xml.

Forma el id de la regla añadiendo a “R” el contador de las reglas. Añade al mapa de ruleIds la regla con el id de la regla como clave. Crea la etiqueta “regla” y la añade a la etiqueta traductor, después le añade un atributo “id” con el id y recoge las acciones semánticas de la regla. Por cada acción semántica:

crea una etiqueta “accionSemantica”, la añade a la etiqueta de regla, obtiene la posición de la acción semántica a la regla para crear un atributo ”pos” con el valor obtenido. Añade como texto la acción semántica y si la acción esta en el medio de la regla se añade una etiqueta “intermedio” con el texto sí.

Añade los símbolos de la producción. Y suma uno al contador de reglas.

**private ArrayList<String> actions(String production):**

Dada una producción devuelve todas las acciones en ella.

Separa la producción en símbolos y por cada símbolo si es una acción la añade a la lista que va a devolver.

**private Integer getPos(String action, String production):**

Dada una acción y una producción devuelve la posición detrás de la que se encuentra la acción.

Divide la producción en símbolos y mientras no sean igual a la acción si el símbolo no es una acción se suma uno a la posición y se pasa al siguiente símbolo.

**private void addSymbols(String production, Element regla,String antecedente):**

Dada una producción, un antecedente y la etiqueta regla escribe los símbolos de la producción en el xml.

Separa la producción en símbolos, por cada símbolo:

Si es el primero crea la etiqueta “simbolo” , la añade a regla , crea la etiqueta “valor” y la añade a símbolo. Si es la primera producción de la regla le añade a valor el antecedente si no añade “|”. Por último, crea la etiqueta “terminal”, la añade a símbolo y le añade el texto “false”.

Si el símbolo no es una acción crea la etiqueta “simbolo” , la añade a regla , crea la etiqueta “valor” y la añade a símbolo. Le añade a valor el símbolo. Crea la etiqueta “terminal”, la añade a símbolo y le añade el texto “false” si la primera letra es mayúscula y “true” si no.

**private void writeArbol(Integer altura):**

Escribe la parte de <arbol> del xml.

Crea la etiqueta árbol y la añade al documento. Crea la etiqueta numNodos y la añade la etiqueta arbol con el número de nodos como texto. Crea la etiqueta altura y la añade la etiqueta arbol con la altura del árbol como texto. Por cada nodo en la lista de nodos:

Creo la etiqueta nodo y la añado a la etiqueta arbol con un atributo con el id del nodo. Creo la etiqueta elemento y la añado a la etiqueta nodo con el elemento del nodo como texto, creo la etiqueta nivel y la añado a la etiqueta nodo con el nivel del nodo como texto, creo la etiqueta terminal y la añado a la etiqueta nodo con true si es un terminal y false si no como texto.

**private void writeContenido():**

Escribe la parte de <contenido> del xml.

Crea la etiqueta contenido y la añade al documento. Por cada paso de la lista pasos:

Crea la etiqueta paso y la añade a contenido con un atributo id. Crea la etiqueta tipo y la añade a contenido con el texto del tipo de paso, Si el paso tiene regla:

Crea la etiqueta nuevaRegla y la añade a paso con un atributo refRegla con el id de la regla y la regla como texto. Crea la etiqueta widthRegla y la añade a paso con el ancho de la regla como texto.

Crea la etiqueta cadena y la añade a paso. Crea la etiqueta leido y la añade a cadena con la cadena leído como texto. Crea la etiqueta pendiente y la añade a cadena con la cadena pendiente como texto. Crea la etiqueta elemento y la añade a paso con el elemento del paso como texto. Crea la etiqueta valor y la añade a paso con el valor del paso como texto. Si tiene nodos que cambien de valor por él:

Crea la etiqueta accionSemanticaEjecutada y la añade a paso. Por cada nodo que cambia de valor:

Crea la etiqueta nodo y la añade a accionSemanticaEjecutada. Crea la etiqueta refNodo y la añade a nodo con el id del node que cambia. Crea la etiqueta atributos y la añade a nodo con el valor al que cambia el nodo.

# ApiCreatorXML V0.2

## Clase ApiCreatorXML V0.1

Se utiliza para comprobar el funcionamiento de la API. Crea el escritor y escribe la parte del traductor. Añade los nodos y escribe la parte del árbol. Añade una serie de pasos y escribe la parte de contenido. Por último, crea el XML.

## Clase Node V0.2

Representa los nodos de la parte de árbol del XML

**Atributos:**

**private Integer id:** id del nodo, se crea automáticamente según vas añadiendo nodos.

**private String element:** elemento del nodo.

**private Boolean terminal:** true si es terminal false si no lo es.

**private Integer nivel:** niveldel árbol en el que se encuentra.

**Private Node fahterNode:** nodo padre en el árbol sintáctico.

**Constructor:**

**public Node(Integer id, String element, Boolean terminal, Integer nivel):**

Recoge los datos pertenecientes a todos los atributos y los asigna.

## Clase Paso V0.2

**Atributos:**

**private Integer id:** representa el id del paso, el cual se asigna automáticamente cuando se añaden los pasos.

**private String tipo:** describe el tipo de paso(“DespDes” para desplazamientos en descendentes, “derivacion” para derivaciones, “desplazamiento” para desplazamientos en ascendentes y “reduccion” para reducciones ).

**private String leído:** parte de la cadena leída.

**private String pendiente:** parte dela cadena pendiente.

**private String elemento:** elemento que se está procesando.

**private String valor:** valor del elemento que se está procesando.

**private String regla:** regla correspondiente al paso (dicha regla solo se añade si es el primer elemento de la regla )

**private HashSet<Integer> relNodo:** nodo que produce este paso. (paso padre).

**private ArrayList<Integer> changedNodes:** nodos que cambian después de este paso.

**private ArrayList<String> changes:** valor de los pasos que van a cambiar después de este paso.

**Constructor:**

**public Paso(Integer id, String tipo, String leido, String pendiente, String elemento, String valor,** **HashSet<Integer>, String regla)**

Recoge los valores de los atributos e inicializa las listas **changedNodes** y **changes.**

## Clase Writer V0.2

**Atributos:**

**private HashMap<String, ArrayList<String>> grammar:** corresponde a la gramática , es un mapa que contiene las producciones por cada antecedente.

**private String path:** ruta de la gramática.

**private String pathResult:** ruta donde guardar el xml.

**private Stack<String> pendChain:** pila de String que representan la cadena que aun no se ha procesado.

**private String readChain:** String que representa la parte de la cadena leída.

**private Element espec:** corresponde al elemento a partir del que se va a escribir las partes del traductor , la cadena , el árbol y el contenido.

**private Document doc:** representa el documento XML.

**private Integer ruleCount:** contador de reglas.

**private ArrayList<String> antecedentes:** lista con los antecedentes de la gramática.

**private Integer numNodos:** contador de nodos.

**private ArrayList<Node> nodes:** lista de nodos.

**private ArrayList<Paso> steps:** lista de pasos.

**private HashMap<String, String> ruleId:** mapa con los ids de cada regla.

**private Integer pasoCount:** contador de pasos.

**private String traductorType:** representa el tipo de traductor(Ascendente o Descendente) **.**

**Constructor:**

**public Writer( String path, String pathResult, String entryChainPath, Boolean isDescendat):**

Recoge la ruta de la gramática, la ruta de destino y la ruta de la cadena de entrada y si es descendente (true) o ascendente (false).

Inicializa los mapas de la gramática y de los ids por regla y las listas de antecedentes, pasos y nodos. Lee la gramática y la guarda en los mapas. Inicializa la pila de cadena pendiente y lee la cadena del archivo. Si es descendente(true) de le asigna “Descendente” al tipo de traductor y si es ascendente(false) “Ascendente”. Inicializa el documento XML y escribe la parte de <traductor> del XML.

**Métodos:**

**public Node addNode(String element, Boolean terminal, Integer nivel):**

Crea un nodo, lo añade a la lista de nodos y aumenta el contador de nodos. Devuelve el nodo creado.

**public Paso addPaso(Boolean isDisplacement, String elementoLeido, String element, String valor, String regla,HashSet<Integer> relNodo):**

Recoge si un paso es un desplazamiento, el elemento de la cadena que se haya podido leer en ese paso, el elemento procesado en ese paso , el valor del paso , la regla que puede tener ese paso y los nodos con los que se relaciona.

Si se ha procesado un terminal se saca de la cadena de pendientes. Se convierte la cadena de pendientes a String.

Si el paso era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “despDes”. Crea un paso.

Si el paso era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “desplazamineto”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “derivacion”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “reduccion”. Crea un paso.

Añade a la lista de pasos el paso creado y aumenta el contador de pasos.

Devuelve el paso creado.

**public Paso addPaso(Boolean isDisplacement,String elementoLeido,String element, String valor, String regla):**

Recoge si un paso es un desplazamiento, el elemento de la cadena que se haya podido leer en ese paso, el elemento procesado en ese paso , el valor del paso y la regla que puede tener ese paso. Se usa para nodos terminales en el traductor ascendente.

Si se ha procesado un terminal se saca de la cadena de pendientes. Se convierte la cadena de pendientes a String.

Si el paso era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “despDes”. Crea un paso.

Si el paso era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “desplazamineto”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “derivacion”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “reduccion”. Crea un paso.

Añade a la lista de pasos el paso creado y aumenta el contador de pasos.

Devuelve el paso creado.

**public Paso addPasoPrimero( String element, String valor, String regla):**

Recoge el elemento procesado en ese paso , el valor del paso y la regla que puede tener ese paso. Se usa para el primer nodo en un traductor descendente.

Se convierte la cadena de pendientes a String.

Crea un paso. Con el tipo de paso “primero”.

Añade a la lista de pasos el paso creado y aumenta el contador de pasos.

Devuelve el paso creado.

**public Paso addPaso(Boolean isDisplacement,String elementoLeido,String element, String valor, String regla,Integer relNodo):**

Recoge si un paso es un desplazamiento, el elemento de la cadena que se haya podido leer en ese paso, el elemento procesado en ese paso , el valor del paso, la regla que puede tener ese paso y el nodo con el que se relaciona. Se usa para nodos con un único nodo relacionado como en el caso de los traductores ascendentes.

Crea un conjunto de Integer y añade el nodo relacionado. Si se ha procesado un terminal se saca de la cadena de pendientes. Se convierte la cadena de pendientes a String.

Si el paso era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “despDes”. Crea un paso.

Si el paso era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “desplazamineto”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “derivacion”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “reduccion”. Crea un paso.

Añade a la lista de pasos el paso creado y aumenta el contador de pasos.

Devuelve el paso creado.

**public Boolean writeXML():**

Crea el archivo XML.

**private String removeActions(String production):**

Dada una producción elimina las acciones y los símbolos recursivos.

Se separa la producción y por cada símbolo si no es una acción, si pertenece a los antecedentes se añade al resultado , si no pertenece se añade al resultado la parte del símbolo que termina cuando empiezan los dígitos.

**private String removeOnlyActions(String production):**

Dada una producción elimina las acciones.

Se separa la producción y por cada símbolo si no es una acción se añade al resultado.

**private Integer getNumberIndex(String symbol):**

Devuelve el índice donde empiezan los dígitos en un símbolo recursivo.

Por cada letra se suma uno hasta que se detecta un digito.

**private void readFile():**

Lee y guarda la gramática desde un archivo de texto.

Por cada línea la separa en antecedente y producciones y se separa las producciones. Se añade el antecedente a la lista si no está y se guarda en el mapa de la gramática con acciones una lista nueva con el antecedente como clave. Se añaden cada producción a la lista anteriormente mencionada.

**public void writeTraductor():**

Escribe la parte de <traductor> y <cadena> del XML.

Se crea la etiqueta “traductor” y se añade a la etiqueta espec. Se crea la etiqueta “tipo” y se añade a traductor, después se añade el texto “Descendente” a tipo. Por cada antecedente , por cada producción se añade la regla. Cuando se han añadido todas las reglas se crea la etiqueta “cadena” y se añade a espec, por último, se añade el texto de la cadena de entrada a cadena.

**private void addRule(String antecedent ,String production,Element traductor):**

Dado un antecedente , una producción y la etiqueta a la que añadir la regla , añade una regla al xml.

Forma el id de la regla añadiendo a “R” el contador de las reglas. Añade al mapa de ruleIds la regla con el id de la regla como clave. Crea la etiqueta “regla” y la añade a la etiqueta traductor, después le añade un atributo “id” con el id y recoge las acciones semánticas de la regla. Por cada acción semántica:

crea una etiqueta “accionSemantica”, la añade a la etiqueta de regla, obtiene la posición de la acción semántica a la regla para crear un atributo ”pos” con el valor obtenido. Añade como texto la acción semántica y si la acción está en el medio de la regla se añade una etiqueta “intermedio” con el texto sí.

Añade los símbolos de la producción. Y suma uno al contador de reglas.

**private ArrayList<String> actions(String production):**

Dada una producción devuelve todas las acciones en ella.

Separa la producción en símbolos y por cada símbolo si es una acción la añade a la lista que va a devolver.

**private Integer getPos(String action, String production):**

Dada una acción y una producción devuelve la posición detrás de la que se encuentra la acción.

Divide la producción en símbolos y mientras no sean igual a la acción si el símbolo no es una acción se suma uno a la posición y se pasa al siguiente símbolo.

**private void addSymbols(String production, Element regla,String antecedente):**

Dada una producción, un antecedente y la etiqueta regla escribe los símbolos de la producción en el xml.

Separa la producción en símbolos, por cada símbolo:

Si es el primero crea la etiqueta “simbolo” , la añade a regla , crea la etiqueta “valor” y la añade a símbolo. Si es la primera producción de la regla le añade a valor el antecedente si no añade “|”. Por último, crea la etiqueta “terminal”, la añade a símbolo y le añade el texto “false”.

Si el símbolo no es una acción crea la etiqueta “simbolo” , la añade a regla , crea la etiqueta “valor” y la añade a símbolo. Le añade a valor el símbolo. Crea la etiqueta “terminal”, la añade a símbolo y le añade el texto “false” si la primera letra es mayúscula y “true” si no.

**public void writeArbol():**

Escribe la parte de <arbol> del xml.

Actualiza el nivel de los nodos y recoge la altura del árbol. Crea la etiqueta árbol y la añade al documento. Crea la etiqueta numNodos y la añade la etiqueta arbol con el número de nodos como texto. Crea la etiqueta altura y la añade la etiqueta arbol con la altura del árbol como texto. Por cada nodo en la lista de nodos:

Creo la etiqueta nodo y la añado a la etiqueta arbol con un atributo con el id del nodo. Creo la etiqueta elemento y la añado a la etiqueta nodo con el elemento del nodo como texto, creo la etiqueta nivel y la añado a la etiqueta nodo con el nivel del nodo como texto, creo la etiqueta terminal y la añado a la etiqueta nodo con true si es un terminal y false si no como texto.

**public void writeContenido():**

Escribe la parte de <contenido> del xml.

Crea la etiqueta contenido y la añade al documento. Por cada paso de la lista pasos:

Crea la etiqueta paso y la añade a contenido con un atributo id. Crea la etiqueta tipo y la añade a contenido con el texto del tipo de paso, Si el paso tiene regla:

Crea la etiqueta nuevaRegla y la añade a paso con un atributo refRegla con el id de la regla y la regla como texto. Crea la etiqueta widthRegla y la añade a paso con el ancho de la regla como texto.

Crea la etiqueta cadena y la añade a paso. Crea la etiqueta leido y la añade a cadena con la cadena leído como texto. Crea la etiqueta pendiente y la añade a cadena con la cadena pendiente como texto. Crea la etiqueta elemento y la añade a paso con el elemento del paso como texto. Si tiene nodos relacionados crea la etiqueta relNodos y la añade a paso con los ids de los nodos relacionados del paso como texto. Crea la etiqueta valor y la añade a paso con el valor del paso como texto. Si tiene nodos que cambien de valor por él:

Crea la etiqueta accionSemanticaEjecutada y la añade a paso. Por cada nodo que cambia de valor:

Crea la etiqueta nodo y la añade a accionSemanticaEjecutada. Crea la etiqueta refNodo y la añade a nodo con el id del nodo que cambia. Crea la etiqueta atributos y la añade a nodo con el valor al que cambia el nodo.

**private String writeRelNodes(HashSet<Integer> relNodo):**

Convierte un conjunto de ids de nodos en una String.

**private void readChain(String entryChainPath):**

Lee la cadena de entrada de un archivo.

Por cada línea del archivo por cada elemento de la cadena (tiene que estar separados por un espacio), lo añade a la pila de cadena pendiente.

**private String writePendChain():**

Transforma la pila de cadena pendiente en un String.

**private Integer updateNode():**

Actualiza el nivel de los nodos.

Recoge la raíz del árbol. Por cada nodo:

Calcula la distancia desde el nodo a la raíz, actualiza el nodo, se queda con el nivel más alto.

Devuelve la altura del árbol sintáctico.

**private Integer distanciaARaiz(Paso raiz, Node objetivo):**

Cuento los nodos padres hasta llegar a la raíz. Devuelve la cantidad de nodos padre.

# ApiCreatorXML V0.3

## Clase ApiCreatorXML V0.1

Se utiliza para comprobar el funcionamiento de la API. Crea el escritor y escribe la parte del traductor. Añade los nodos y escribe la parte del árbol. Añade una serie de pasos y escribe la parte de contenido. Por último, crea el XML.

## Clase Node V0.3

Representa los nodos de la parte de árbol del XML

**Atributos:**

**private Integer id:** id del nodo, se crea automáticamente según vas añadiendo nodos.

**private String element:** elemento del nodo.

**private Boolean terminal:** true si es terminal false si no lo es.

**private Integer nivel:** niveldel árbol en el que se encuentra.

**private Node fahterNode:** nodo padre en el árbol sintáctico.

**private Boolean haveBrother:** true si el nodo tiene hermano derecho y false si no. Este atributo esta pensado para usarse en los traductores descendentes para saber cuando parar de actualizar valores.

**Constructor:**

**public Node(Integer id, String element, Boolean terminal, Integer nivel** **, Boolean haveBrother):**

Recoge los datos pertenecientes a todos los atributos y los asigna.

## Clase Paso V0.2

**Atributos:**

**private Integer id:** representa el id del paso, el cual se asigna automáticamente cuando se añaden los pasos.

**private String tipo:** describe el tipo de paso(“DespDes” para desplazamientos en descendentes, “derivacion” para derivaciones, “desplazamiento” para desplazamientos en ascendentes y “reduccion” para reducciones ).

**private String leído:** parte de la cadena leída.

**private String pendiente:** parte dela cadena pendiente.

**private String elemento:** elemento que se está procesando.

**private String valor:** valor del elemento que se está procesando.

**private String regla:** regla correspondiente al paso (dicha regla solo se añade si es el primer elemento de la regla )

**private HashSet<Integer> relNodo:** nodo que produce este paso. (paso padre).

**private ArrayList<Integer> changedNodes:** nodos que cambian después de este paso.

**private ArrayList<String> changes:** valor de los pasos que van a cambiar después de este paso.

**Constructor:**

**public Paso(Integer id, String tipo, String leido, String pendiente, String elemento, String valor,** **HashSet<Integer>, String regla)**

Recoge los valores de los atributos e inicializa las listas **changedNodes** y **changes.**

## Clase Writer V0.3

**Atributos:**

**private HashMap<String, ArrayList<String>> grammar:** corresponde a la gramática , es un mapa que contiene las producciones por cada antecedente.

**private String path:** ruta de la gramática.

**private String pathResult:** ruta donde guardar el xml.

**private Stack<String> pendChain:** pila de String que representan la cadena que aún no se ha procesado.

**private String readChain:** String que representa la parte de la cadena leída.

**private Element espec:** corresponde al elemento a partir del que se va a escribir las partes del traductor , la cadena , el árbol y el contenido.

**private Document doc:** representa el documento XML.

**private Integer ruleCount:** contador de reglas.

**private ArrayList<String> antecedentes:** lista con los antecedentes de la gramática.

**private Integer numNodos:** contador de nodos.

**private ArrayList<Node> nodes:** lista de nodos.

**private ArrayList<Paso> steps:** lista de pasos.

**private HashMap<String, String> ruleId:** mapa con los ids de cada regla.

**private Integer pasoCount:** contador de pasos.

**private String traductorType:** representa el tipo de traductor(Ascendente o Descendente) **.**

**private HashMap<Integer,Paso>stepMaps:** mapa que guarda los pasos por id para poder recuperarlos de una forma rápida y así actualizar los valores del paso.

**Constructor:**

**public** **Writer( String path, String pathResult, String entryChainPath, Boolean isDescendat):**

Recoge la ruta de la gramática, la ruta de destino y la ruta de la cadena de entrada y si es descendente (true) o ascendente (false).

Inicializa los mapas de la gramática ,de los ids por regla y de los pasos por id y las listas de antecedentes, pasos y nodos. Lee la gramática y la guarda en los mapas. Inicializa la pila de cadena pendiente y lee la cadena del archivo. Si es descendente(true) de le asigna “Descendente” al tipo de traductor y si es ascendente(false) “Ascendente”. Inicializa el documento XML y escribe la parte de <traductor> del XML.

**Métodos:**

**public Node addNode(String element, Boolean terminal, Integer nivel** **,Boolean haveBrother):**

Crea un nodo, lo añade a la lista de nodos y aumenta el contador de nodos. Devuelve el nodo creado.

**public Paso addPaso(Boolean isDisplacement, String elementoLeido, String element, String valor, String regla,HashSet<Integer> relNodo):**

Recoge si un paso es un desplazamiento, el elemento de la cadena que se haya podido leer en ese paso, el elemento procesado en ese paso , el valor del paso , la regla que puede tener ese paso y los nodos con los que se relaciona.

Si se ha procesado un terminal se saca de la cadena de pendientes. Se convierte la cadena de pendientes a String.

Si el paso era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “despDes”. Crea un paso.

Si el paso era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “desplazamineto”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “derivacion”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “reduccion”. Crea un paso.

Añade a la lista de pasos y al mapa el paso creado y aumenta el contador de pasos.

Devuelve el paso creado.

**public Paso addPaso(Boolean isDisplacement,String elementoLeido,String element, String valor, String regla):**

Recoge si un paso es un desplazamiento, el elemento de la cadena que se haya podido leer en ese paso, el elemento procesado en ese paso , el valor del paso y la regla que puede tener ese paso. Se usa para nodos terminales en el traductor ascendente.

Si se ha procesado un terminal se saca de la cadena de pendientes. Se convierte la cadena de pendientes a String.

Si el paso era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “despDes”. Crea un paso.

Si el paso era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “desplazamineto”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “derivacion”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “reduccion”. Crea un paso.

Añade a la lista de pasos y al mapa el paso creado y aumenta el contador de pasos.

Devuelve el paso creado.

**public Paso addPasoPrimero( String element, String valor, String regla):**

Recoge el elemento procesado en ese paso , el valor del paso y la regla que puede tener ese paso. Se usa para el primer nodo en un traductor descendente.

Se convierte la cadena de pendientes a String.

Crea un paso. Con el tipo de paso “primero”.

Añade a la lista de pasos y al mapa el paso creado y aumenta el contador de pasos.

Devuelve el paso creado.

**public Paso addPaso(Boolean isDisplacement,String elementoLeido,String element, String valor, String regla,Integer relNodo):**

Recoge si un paso es un desplazamiento, el elemento de la cadena que se haya podido leer en ese paso, el elemento procesado en ese paso , el valor del paso, la regla que puede tener ese paso y el nodo con el que se relaciona. Se usa para nodos con un único nodo relacionado como en el caso de los traductores ascendentes.

Crea un conjunto de Integer y añade el nodo relacionado. Si se ha procesado un terminal se saca de la cadena de pendientes. Se convierte la cadena de pendientes a String.

Si el paso era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “despDes”. Crea un paso.

Si el paso era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “desplazamineto”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “derivacion”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “reduccion”. Crea un paso.

Añade a la lista de pasos y al mapa el paso creado y aumenta el contador de pasos.

Devuelve el paso creado.

**public Boolean writeXML():**

Escribe la parte de <contenido> y la parte de <arbol> del xml. Después crea el archivo XML.

**private String removeActions(String production):**

Dada una producción elimina las acciones y los símbolos recursivos.

Se separa la producción y por cada símbolo si no es una acción, si pertenece a los antecedentes se añade al resultado , si no pertenece se añade al resultado la parte del símbolo que termina cuando empiezan los dígitos.

**private String removeOnlyActions(String production):**

Dada una producción elimina las acciones.

Se separa la producción y por cada símbolo si no es una acción se añade al resultado.

**private Integer getNumberIndex(String symbol):**

Devuelve el índice donde empiezan los dígitos en un símbolo recursivo.

Por cada letra se suma uno hasta que se detecta un digito.

**private void readFile():**

Lee y guarda la gramática desde un archivo de texto.

Por cada línea la separa en antecedente y producciones y se separa las producciones. Se añade el antecedente a la lista si no está y se guarda en el mapa de la gramática con acciones una lista nueva con el antecedente como clave. Se añaden cada producción a la lista anteriormente mencionada.

**private void writeTraductor():**

Escribe la parte de <traductor> y <cadena> del XML.

Se crea la etiqueta “traductor” y se añade a la etiqueta espec. Se crea la etiqueta “tipo” y se añade a traductor, después se añade el texto “Descendente” a tipo. Por cada antecedente , por cada producción se añade la regla. Cuando se han añadido todas las reglas se crea la etiqueta “cadena” y se añade a espec, por último, se añade el texto de la cadena de entrada a cadena.

**private void addRule(String antecedent ,String production,Element traductor):**

Dado un antecedente , una producción y la etiqueta a la que añadir la regla , añade una regla al xml.

Forma el id de la regla añadiendo a “R” el contador de las reglas. Añade al mapa de ruleIds la regla con el id de la regla como clave. Crea la etiqueta “regla” y la añade a la etiqueta traductor, después le añade un atributo “id” con el id y recoge las acciones semánticas de la regla. Por cada acción semántica:

crea una etiqueta “accionSemantica”, la añade a la etiqueta de regla, obtiene la posición de la acción semántica a la regla para crear un atributo ”pos” con el valor obtenido. Añade como texto la acción semántica y si la acción está en el medio de la regla se añade una etiqueta “intermedio” con el texto sí.

Añade los símbolos de la producción. Y suma uno al contador de reglas.

**private ArrayList<String> actions(String production):**

Dada una producción devuelve todas las acciones en ella.

Separa la producción en símbolos y por cada símbolo si es una acción la añade a la lista que va a devolver.

**private Integer getPos(String action, String production):**

Dada una acción y una producción devuelve la posición detrás de la que se encuentra la acción.

Divide la producción en símbolos y mientras no sean igual a la acción si el símbolo no es una acción se suma uno a la posición y se pasa al siguiente símbolo.

**private void addSymbols(String production, Element regla,String antecedente):**

Dada una producción, un antecedente y la etiqueta regla escribe los símbolos de la producción en el xml.

Separa la producción en símbolos, por cada símbolo:

Si es el primero crea la etiqueta “simbolo” , la añade a regla , crea la etiqueta “valor” y la añade a símbolo. Si es la primera producción de la regla le añade a valor el antecedente si no añade “|”. Por último, crea la etiqueta “terminal”, la añade a símbolo y le añade el texto “false”.

Si el símbolo no es una acción crea la etiqueta “simbolo” , la añade a regla , crea la etiqueta “valor” y la añade a símbolo. Le añade a valor el símbolo. Crea la etiqueta “terminal”, la añade a símbolo y le añade el texto “false” si la primera letra es mayúscula y “true” si no.

private void writeArbol():

Escribe la parte de <arbol> del xml.

Actualiza el nivel de los nodos y recoge la altura del árbol. Crea la etiqueta árbol y la añade al documento. Crea la etiqueta numNodos y la añade la etiqueta arbol con el número de nodos como texto. Crea la etiqueta altura y la añade la etiqueta arbol con la altura del árbol como texto. Por cada nodo en la lista de nodos:

Creo la etiqueta nodo y la añado a la etiqueta arbol con un atributo con el id del nodo. Creo la etiqueta elemento y la añado a la etiqueta nodo con el elemento del nodo como texto, creo la etiqueta nivel y la añado a la etiqueta nodo con el nivel del nodo como texto, creo la etiqueta terminal y la añado a la etiqueta nodo con true si es un terminal y false si no como texto.

private void writeContenido():

Escribe la parte de <contenido> del xml.

Crea la etiqueta contenido y la añade al documento. Por cada paso de la lista pasos:

Crea la etiqueta paso y la añade a contenido con un atributo id. Crea la etiqueta tipo y la añade a contenido con el texto del tipo de paso, Si el paso tiene regla:

Crea la etiqueta nuevaRegla y la añade a paso con un atributo refRegla con el id de la regla y la regla como texto. Crea la etiqueta widthRegla y la añade a paso con el ancho de la regla como texto.

Crea la etiqueta cadena y la añade a paso. Crea la etiqueta leido y la añade a cadena con la cadena leído como texto. Crea la etiqueta pendiente y la añade a cadena con la cadena pendiente como texto. Crea la etiqueta elemento y la añade a paso con el elemento del paso como texto. Si tiene nodos relacionados crea la etiqueta relNodos y la añade a paso con los ids de los nodos relacionados del paso como texto. Crea la etiqueta valor y la añade a paso con el valor del paso como texto. Si tiene nodos que cambien de valor por él:

Crea la etiqueta accionSemanticaEjecutada y la añade a paso. Por cada nodo que cambia de valor:

Crea la etiqueta nodo y la añade a accionSemanticaEjecutada. Crea la etiqueta refNodo y la añade a nodo con el id del nodo que cambia. Crea la etiqueta atributos y la añade a nodo con el valor al que cambia el nodo.

**private String writeRelNodes(HashSet<Integer> relNodo):**

Convierte un conjunto de ids de nodos en una String.

**private void readChain(String entryChainPath):**

Lee la cadena de entrada de un archivo.

Por cada línea del archivo por cada elemento de la cadena (tiene que estar separados por un espacio), lo añade a la pila de cadena pendiente.

**private String writePendChain():**

Transforma la pila de cadena pendiente en un String.

**private Integer updateNode():**

Actualiza el nivel de los nodos.

Recoge la raíz del árbol. Por cada nodo:

Calcula la distancia desde el nodo a la raíz, actualiza el nodo, se queda con el nivel más alto.

Devuelve la altura del árbol sintáctico.

**private Integer distanciaARaiz(Paso raiz, Node objetivo):**

Cuento los nodos padres hasta llegar a la raíz. Devuelve la cantidad de nodos padre.

**public Paso getStep(Integer id):**

Devuelve el paso que corresponda con el id dado.

# ApiCreatorXML V0.4

## Clase ApiCreatorXML V0.1

Se utiliza para comprobar el funcionamiento de la API. Crea el escritor y escribe la parte del traductor. Añade los nodos y escribe la parte del árbol. Añade una serie de pasos y escribe la parte de contenido. Por último, crea el XML.

## Clase Node V0.3

Representa los nodos de la parte de árbol del XML

**Atributos:**

**private Integer id:** id del nodo, se crea automáticamente según vas añadiendo nodos.

**private String element:** elemento del nodo.

**private Boolean terminal:** true si es terminal false si no lo es.

**private Integer nivel:** niveldel árbol en el que se encuentra.

**private Node fahterNode:** nodo padre en el árbol sintáctico.

**private Boolean haveBrother:** true si el nodo tiene hermano derecho y false si no. Este atributo esta pensado para usarse en los traductores descendentes para saber cuando parar de actualizar valores.

**Constructor:**

**public Node(Integer id, String element, Boolean terminal, Integer nivel** **, Boolean haveBrother):**

Recoge los datos pertenecientes a todos los atributos y los asigna.

## Clase Paso V0.2

**Atributos:**

**private Integer id:** representa el id del paso, el cual se asigna automáticamente cuando se añaden los pasos.

**private String tipo:** describe el tipo de paso(“DespDes” para desplazamientos en descendentes, “derivacion” para derivaciones, “desplazamiento” para desplazamientos en ascendentes y “reduccion” para reducciones ).

**private String leído:** parte de la cadena leída.

**private String pendiente:** parte dela cadena pendiente.

**private String elemento:** elemento que se está procesando.

**private String valor:** valor del elemento que se está procesando.

**private String regla:** regla correspondiente al paso (dicha regla solo se añade si es el primer elemento de la regla )

**private HashSet<Integer> relNodo:** nodo que produce este paso. (paso padre).

**private ArrayList<Integer> changedNodes:** nodos que cambian después de este paso.

**private ArrayList<String> changes:** valor de los pasos que van a cambiar después de este paso.

**Constructor:**

**public Paso(Integer id, String tipo, String leido, String pendiente, String elemento, String valor,** **HashSet<Integer>, String regla)**

Recoge los valores de los atributos e inicializa las listas **changedNodes** y **changes.**

## Clase Writer V0.4

**Atributos:**

**private HashMap<String, ArrayList<String>> grammar:** corresponde a la gramática , es un mapa que contiene las producciones por cada antecedente.

**private String path:** ruta de la gramática.

**private String pathResult:** ruta donde guardar el xml.

**private Stack<String> pendChain:** pila de String que representan la cadena que aún no se ha procesado.

**private String readChain:** String que representa la parte de la cadena leída.

**private Element espec:** corresponde al elemento a partir del que se va a escribir las partes del traductor , la cadena , el árbol y el contenido.

**private Document doc:** representa el documento XML.

**private Integer ruleCount:** contador de reglas.

**private ArrayList<String> antecedentes:** lista con los antecedentes de la gramática.

**private Integer numNodos:** contador de nodos.

**private ArrayList<Node> nodes:** lista de nodos.

**private ArrayList<Paso> steps:** lista de pasos.

**private HashMap<String, String> ruleId:** mapa con los ids de cada regla.

**private Integer pasoCount:** contador de pasos.

**private String traductorType:** representa el tipo de traductor(Ascendente o Descendente) **.**

**private HashMap<Integer,Paso>stepMaps:** mapa que guarda los pasos por id para poder recuperarlos de una forma rápida y así actualizar los valores del paso.

**Constructor:**

**public Writer( String path, String pathResult, String entryChainPath, Boolean isDescendat):**

Recoge la ruta de la gramática, la ruta de destino y la ruta de la cadena de entrada y si es descendente (true) o ascendente (false).

Inicializa los mapas de la gramática ,de los ids por regla y de los pasos por id y las listas de antecedentes, pasos y nodos. Lee la gramática y la guarda en los mapas. Inicializa la pila de cadena pendiente y lee la cadena del archivo. Si es descendente(true) de le asigna “Descendente” al tipo de traductor y si es ascendente(false) “Ascendente”. Inicializa el documento XML y escribe la parte de <traductor> del XML.

**Métodos:**

**public Node addNode(String element, Boolean terminal, Integer nivel** **,Boolean haveBrother):**

Crea un nodo, lo añade a la lista de nodos y aumenta el contador de nodos. Devuelve el nodo creado.

**public Paso addPaso(Boolean isDisplacement, String elementoLeido, String element, String valor, String regla,HashSet<Integer> relNodo):**

Recoge si un paso es un desplazamiento, el elemento de la cadena que se haya podido leer en ese paso, el elemento procesado en ese paso , el valor del paso , la regla que puede tener ese paso y los nodos con los que se relaciona.

Si se ha procesado un terminal se saca de la cadena de pendientes. Se convierte la cadena de pendientes a String.

Si el paso era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “despDes”. Crea un paso.

Si el paso era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “desplazamineto”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “derivacion”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “reduccion”. Crea un paso.

Añade a la lista de pasos y al mapa el paso creado y aumenta el contador de pasos.

Devuelve el paso creado.

**public Paso addPaso(Boolean isDisplacement,String elementoLeido,String element, String valor, String regla):**

Recoge si un paso es un desplazamiento, el elemento de la cadena que se haya podido leer en ese paso, el elemento procesado en ese paso , el valor del paso y la regla que puede tener ese paso. Se usa para nodos terminales en el traductor ascendente.

Si se ha procesado un terminal se saca de la cadena de pendientes. Se convierte la cadena de pendientes a String.

Si el paso era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “despDes”. Crea un paso.

Si el paso era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “desplazamineto”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “derivacion”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “reduccion”. Crea un paso.

Añade a la lista de pasos y al mapa el paso creado y aumenta el contador de pasos.

Devuelve el paso creado.

**public Paso addPasoPrimero( String element, String valor, String regla):**

Recoge el elemento procesado en ese paso , el valor del paso y la regla que puede tener ese paso. Se usa para el primer nodo en un traductor descendente.

Se convierte la cadena de pendientes a String.

Crea un paso. Con el tipo de paso “primero”.

Añade a la lista de pasos y al mapa el paso creado y aumenta el contador de pasos.

Devuelve el paso creado.

**public Paso addPaso(Boolean isDisplacement,String elementoLeido,String element, String valor, String regla,Integer relNodo):**

Recoge si un paso es un desplazamiento, el elemento de la cadena que se haya podido leer en ese paso, el elemento procesado en ese paso , el valor del paso, la regla que puede tener ese paso y el nodo con el que se relaciona. Se usa para nodos con un único nodo relacionado como en el caso de los traductores ascendentes.

Crea un conjunto de Integer y añade el nodo relacionado. Si se ha procesado un terminal se saca de la cadena de pendientes. Se convierte la cadena de pendientes a String.

Si el paso era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “despDes”. Crea un paso.

Si el paso era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “desplazamineto”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “derivacion”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “reduccion”. Crea un paso.

Añade a la lista de pasos y al mapa el paso creado y aumenta el contador de pasos.

Devuelve el paso creado.

**public Boolean writeXML():**

Escribe la parte de <contenido> y la parte de <arbol> del xml. Después crea el archivo XML.

**private String removeActions(String production):**

Dada una producción elimina las acciones y los símbolos recursivos.

Se separa la producción y por cada símbolo si no es una acción, si pertenece a los antecedentes se añade al resultado , si no pertenece se añade al resultado la parte del símbolo que termina cuando empiezan los dígitos.

**private String removeOnlyActions(String production):**

Dada una producción elimina las acciones.

Se separa la producción y por cada símbolo si no es una acción se añade al resultado.

**private Integer getNumberIndex(String symbol):**

Devuelve el índice donde empiezan los dígitos en un símbolo recursivo.

Por cada letra se suma uno hasta que se detecta un digito.

**private void readFile():**

Lee y guarda la gramática desde un archivo de texto.

Por cada línea la separa en antecedente y producciones y se separa las producciones. Se añade el antecedente a la lista si no está y se guarda en el mapa de la gramática con acciones una lista nueva con el antecedente como clave. Se añaden cada producción a la lista anteriormente mencionada.

**private void writeTraductor():**

Escribe la parte de <traductor> y <cadena> del XML.

Se crea la etiqueta “traductor” y se añade a la etiqueta espec. Se crea la etiqueta “tipo” y se añade a traductor, después se añade el texto “Descendente” a tipo. Por cada antecedente , por cada producción se añade la regla. Cuando se han añadido todas las reglas se crea la etiqueta “cadena” y se añade a espec, por último, se añade el texto de la cadena de entrada a cadena.

**private void addRule(String antecedent ,String production,Element traductor):**

Dado un antecedente , una producción y la etiqueta a la que añadir la regla , añade una regla al xml.

Forma el id de la regla añadiendo a “R” el contador de las reglas. Añade al mapa de ruleIds la regla con el id de la regla como clave. Crea la etiqueta “regla” y la añade a la etiqueta traductor, después le añade un atributo “id” con el id y recoge las acciones semánticas de la regla. Por cada acción semántica:

crea una etiqueta “accionSemantica”, la añade a la etiqueta de regla, obtiene la posición de la acción semántica a la regla para crear un atributo ”pos” con el valor obtenido. Añade como texto la acción semántica y si la acción está en el medio de la regla se añade una etiqueta “intermedio” con el texto sí.

Añade los símbolos de la producción. Y suma uno al contador de reglas.

**private ArrayList<String> actions(String production):**

Dada una producción devuelve todas las acciones en ella.

Separa la producción en símbolos y por cada símbolo si es una acción la añade a la lista que va a devolver.

**private Integer getPos(String action, String production):**

Dada una acción y una producción devuelve la posición detrás de la que se encuentra la acción.

Divide la producción en símbolos y mientras no sean igual a la acción si el símbolo no es una acción se suma uno a la posición y se pasa al siguiente símbolo.

**private void addSymbols(String production, Element regla,String antecedente):**

Dada una producción, un antecedente y la etiqueta regla escribe los símbolos de la producción en el xml.

Separa la producción en símbolos, por cada símbolo:

Si es el primero crea la etiqueta “simbolo” , la añade a regla , crea la etiqueta “valor” y la añade a símbolo. Si es la primera producción de la regla le añade a valor el antecedente si no añade “|”. Por último, crea la etiqueta “terminal”, la añade a símbolo y le añade el texto “false”.

Si el símbolo no es una acción crea la etiqueta “simbolo” , la añade a regla , crea la etiqueta “valor” y la añade a símbolo. Le añade a valor el símbolo. Crea la etiqueta “terminal”, la añade a símbolo y le añade el texto “false” si la primera letra es mayúscula y “true” si no.

private void writeArbol():

Escribe la parte de <arbol> del xml.

Actualiza el nivel de los nodos y recoge la altura del árbol. Crea la etiqueta árbol y la añade al documento. Crea la etiqueta numNodos y la añade la etiqueta arbol con el número de nodos como texto. Crea la etiqueta altura y la añade la etiqueta arbol con la altura del árbol como texto. Por cada nodo en la lista de nodos:

Creo la etiqueta nodo y la añado a la etiqueta arbol con un atributo con el id del nodo. Creo la etiqueta elemento y la añado a la etiqueta nodo con el elemento del nodo como texto, creo la etiqueta nivel y la añado a la etiqueta nodo con el nivel del nodo como texto, creo la etiqueta terminal y la añado a la etiqueta nodo con true si es un terminal y false si no como texto.

**private void writeContenido():**

Escribe la parte de <contenido> del xml.

Crea la etiqueta contenido y la añade al documento. Por cada paso de la lista pasos:

Crea la etiqueta paso y la añade a contenido con un atributo id. Crea la etiqueta tipo y la añade a contenido con el texto del tipo de paso, Si el paso tiene regla:

Crea la etiqueta nuevaRegla y la añade a paso con un atributo refRegla con el id de la regla y la regla como texto. Crea la etiqueta widthRegla y la añade a paso con el ancho de la regla como texto.

Crea la etiqueta cadena y la añade a paso. Crea la etiqueta leido y la añade a cadena con la cadena leído como texto. Crea la etiqueta pendiente y la añade a cadena con la cadena pendiente como texto. Crea la etiqueta elemento y la añade a paso con el elemento del paso como texto. Si tiene nodos relacionados crea la etiqueta relNodos y la añade a paso con los ids de los nodos relacionados del paso como texto. Crea la etiqueta valor y la añade a paso con el valor del paso como texto. Si tiene nodos que cambien de valor por él:

Crea la etiqueta accionSemanticaEjecutada y la añade a paso. Por cada nodo que cambia de valor:

Crea la etiqueta nodo y la añade a accionSemanticaEjecutada. Crea la etiqueta refNodo y la añade a nodo con el id del nodo que cambia. Crea la etiqueta atributos y la añade a nodo con el valor al que cambia el nodo.

**private String writeRelNodes(HashSet<Integer> relNodo):**

Convierte un conjunto de ids de nodos en una String.

**private void readChain(String entryChainPath):**

Lee la cadena de entrada de un archivo.

Por cada línea del archivo por cada elemento de la cadena (tiene que estar separados por un espacio), lo añade a la pila de cadena pendiente.

**private String writePendChain():**

Transforma la pila de cadena pendiente en un String.

**private Integer updateNode():**

Actualiza el nivel de los nodos.

Recoge la raíz del árbol. Por cada nodo:

Calcula la distancia desde el nodo a la raíz, actualiza el nodo, se queda con el nivel más alto.

Devuelve la altura del árbol sintáctico.

**private Integer distanciaARaiz(Paso raiz, Node objetivo):**

Cuento los nodos padres hasta llegar a la raíz. Devuelve la cantidad de nodos padre.

**public Paso getStep(Integer id):**

Devuelve el paso que corresponda con el id dado.

**public void updatesValues(Paso paso,Node node, String value):**

Recoge y asigna los valores y los nodos que se van a actualizar a este paso. Recoge el padre del nodo introducido y mientras el nodo no tenga hermano derecho añade a la lista de ids de nodos que cambian el id del nodo actual y a la lista de valores el valor cambiado. Para finalizar el bucle cambia el nodo actual por el padre. Para terminar, añade a las actualizaciones el id del nodo que si tiene hermano y el valor actualizado de este valor.

# ApiCreatorXML V0.5

## Clase ApiCreatorXML V0.1

Se utiliza para comprobar el funcionamiento de la API. Crea el escritor y escribe la parte del traductor. Añade los nodos y escribe la parte del árbol. Añade una serie de pasos y escribe la parte de contenido. Por último, crea el XML.

## Clase Node V0.3

Representa los nodos de la parte de árbol del XML

**Atributos:**

**private Integer id:** id del nodo, se crea automáticamente según vas añadiendo nodos.

**private String element:** elemento del nodo.

**private Boolean terminal:** true si es terminal false si no lo es.

**private Integer nivel:** niveldel árbol en el que se encuentra.

**private Node fahterNode:** nodo padre en el árbol sintáctico.

**private Boolean haveBrother:** true si el nodo tiene hermano derecho y false si no. Este atributo esta pensado para usarse en los traductores descendentes para saber cuando parar de actualizar valores.

**Constructor:**

**public Node(Integer id, String element, Boolean terminal, Integer nivel** **, Boolean haveBrother):**

Recoge los datos pertenecientes a todos los atributos y los asigna.

## Clase Paso V0.2

**Atributos:**

**private Integer id:** representa el id del paso, el cual se asigna automáticamente cuando se añaden los pasos.

**private String tipo:** describe el tipo de paso(“DespDes” para desplazamientos en descendentes, “derivacion” para derivaciones, “desplazamiento” para desplazamientos en ascendentes y “reduccion” para reducciones ).

**private String leído:** parte de la cadena leída.

**private String pendiente:** parte dela cadena pendiente.

**private String elemento:** elemento que se está procesando.

**private String valor:** valor del elemento que se está procesando.

**private String regla:** regla correspondiente al paso (dicha regla solo se añade si es el primer elemento de la regla )

**private HashSet<Integer> relNodo:** nodo que produce este paso. (paso padre).

**private ArrayList<Integer> changedNodes:** nodos que cambian después de este paso.

**private ArrayList<String> changes:** valor de los pasos que van a cambiar después de este paso.

**Constructor:**

**public Paso(Integer id, String tipo, String leido, String pendiente, String elemento, String valor,** **HashSet<Integer>, String regla)**

Recoge los valores de los atributos e inicializa las listas **changedNodes** y **changes.**

## Clase Writer V0.5

**Atributos:**

**private HashMap<String, ArrayList<String>> grammar:** corresponde a la gramática , es un mapa que contiene las producciones por cada antecedente.

**private String path:** ruta de la gramática.

**private String pathResult:** ruta donde guardar el xml.

**private Stack<String> pendChain:** pila de String que representan la cadena que aún no se ha procesado.

**private String readChain:** String que representa la parte de la cadena leída.

**private Element espec:** corresponde al elemento a partir del que se va a escribir las partes del traductor , la cadena , el árbol y el contenido.

**private Document doc:** representa el documento XML.

**private Integer ruleCount:** contador de reglas.

**private ArrayList<String> antecedentes:** lista con los antecedentes de la gramática.

**private Integer numNodos:** contador de nodos.

**private ArrayList<Node> nodes:** lista de nodos.

**private ArrayList<Paso> steps:** lista de pasos.

**private HashMap<String, String> ruleId:** mapa con los ids de cada regla.

**private Integer pasoCount:** contador de pasos.

**private String traductorType:** representa el tipo de traductor(Ascendente o Descendente) **.**

**private HashMap<Integer,Paso>stepMaps:** mapa que guarda los pasos por id para poder recuperarlos de una forma rápida y así actualizar los valores del paso.

**Constructor:**

**public Writer( String path, String pathResult, String entryChainPath, Boolean isDescendat):**

Recoge la ruta de la gramática, la ruta de destino y la ruta de la cadena de entrada y si es descendente (true) o ascendente (false).

Inicializa los mapas de la gramática ,de los ids por regla y de los pasos por id y las listas de antecedentes, pasos y nodos. Lee la gramática y la guarda en los mapas. Inicializa la pila de cadena pendiente y lee la cadena del archivo. Si es descendente(true) de le asigna “Descendente” al tipo de traductor y si es ascendente(false) “Ascendente”. Inicializa el documento XML y escribe la parte de <traductor> del XML.

**Métodos:**

**public Node addNode(String element, Boolean terminal, Integer nivel** **,Boolean haveBrother):**

Crea un nodo, lo añade a la lista de nodos y aumenta el contador de nodos. Devuelve el nodo creado.

**public Paso addPaso(Boolean isDisplacement, String elementoLeido, String element, String valor, String regla,HashSet<Integer> relNodo):**

Recoge si un paso es un desplazamiento, el elemento de la cadena que se haya podido leer en ese paso, el elemento procesado en ese paso , el valor del paso , la regla que puede tener ese paso y los nodos con los que se relaciona.

Si se ha procesado un terminal se saca de la cadena de pendientes. Se convierte la cadena de pendientes a String.

Si el paso era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “despDes”. Crea un paso.

Si el paso era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “desplazamineto”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “derivacion”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “reduccion”. Crea un paso.

Añade a la lista de pasos y al mapa el paso creado y aumenta el contador de pasos.

Devuelve el paso creado.

**public Paso addPaso(Boolean isDisplacement,String elementoLeido,String element, String valor, String regla):**

Recoge si un paso es un desplazamiento, el elemento de la cadena que se haya podido leer en ese paso, el elemento procesado en ese paso , el valor del paso y la regla que puede tener ese paso. Se usa para nodos terminales en el traductor ascendente.

Si se ha procesado un terminal se saca de la cadena de pendientes. Se convierte la cadena de pendientes a String.

Si el paso era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “despDes”. Crea un paso.

Si el paso era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “desplazamineto”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “derivacion”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “reduccion”. Crea un paso.

Añade a la lista de pasos y al mapa el paso creado y aumenta el contador de pasos.

Devuelve el paso creado.

**public Paso addPasoPrimero( String element, String valor, String regla):**

Recoge el elemento procesado en ese paso , el valor del paso y la regla que puede tener ese paso. Se usa para el primer nodo en un traductor descendente.

Se convierte la cadena de pendientes a String.

Crea un paso. Con el tipo de paso “primero”.

Añade a la lista de pasos y al mapa el paso creado y aumenta el contador de pasos.

Devuelve el paso creado.

**public Paso addPaso(Boolean isDisplacement,String elementoLeido,String element, String valor, String regla,Integer relNodo):**

Recoge si un paso es un desplazamiento, el elemento de la cadena que se haya podido leer en ese paso, el elemento procesado en ese paso , el valor del paso, la regla que puede tener ese paso y el nodo con el que se relaciona. Se usa para nodos con un único nodo relacionado como en el caso de los traductores ascendentes.

Crea un conjunto de Integer y añade el nodo relacionado. Si se ha procesado un terminal se saca de la cadena de pendientes. Se convierte la cadena de pendientes a String.

Si el paso era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “despDes”. Crea un paso.

Si el paso era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “desplazamineto”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “derivacion”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “reduccion”. Crea un paso.

Añade a la lista de pasos y al mapa el paso creado y aumenta el contador de pasos.

Devuelve el paso creado.

**public Node addPasoLambda(String element, String atributo, Object object):**

Crea una regla con el no terminal y lambda. Añade dos nodos, uno con lambda y otro con el no terminal. Recoge la clase del objeto y el método getValue. Añade dos pasos uno para lambda y otro para el no terminal. Añade el id, el nodo y el paso al no terminal. Por último, devuelve el nodo del no terminal.

**public Node addPasoLambdaDes(String element, String atributoHer, String atributoSint, Object object, Boolean haveBrother, String her, Node nodeAnt, Boolean itsRecursive):**

Crea una regla con el no terminal y lambda. Recoge la clase del objeto. Si el paso es recursivo se le añade un 1 al element. Añade un nodo con el no terminal (con el booleano de si tiene hermano) y un paso uno para el no terminal (con los atributos y her como valor). Añade al nuevo nodo el anterior como padre e introduce uno nuevo con lambda. Añade a este nodo el que hemos creado antes como padre y crea un paso con lambda. Añade el nodo y el paso al no terminal. Por último, actualiza los valores de los pasos conectados y devuelve el nodo del no terminal.

**public Node addPasoNoTerminal(String element, String atributo, Object ... objects):**

Crea la regla con los objetos recogidos y añade un nuevo nodo con el elemento. Por cada objeto empezando por el segundo recoge la clase, el id y el nodo, añade el id a nodos relacionados y el nodo como padre al que hemos creado. Si es el segundo objeto se le añade la regla al paso. Se añade el paso con el no terminal y el valor del primer objeto. Por último, se añade el id del paso creado, el nodo y el paso al primer objeto y devuelve el nodo del no terminal.

**public Node addPasoNoTerminalDes(String element, String atributoHer, String atributoSint, Object object, Boolean haveBrother, String her, Node nodeAnt, Boolean itsRecursive):**

Recoge la clase del objeto. Si es el primer paso añade el nodo y el primer paso. Si no y es un paso recursivo añade el nodo con el elemento mas 1 y el paso con un atributo o con dos si tiene atributo heredado. Si no es recursivo añade el nodo con el elemento y el paso con un atributo o con dos si tiene atributo heredado. Por último, se añade el nodo anterior como padre al creado y el nodo y el paso al objeto, además devuelve el nodo.

**public Node addPasoTerminal(String element, String atributo, Object object):**

Añade un nodo con el terminal. Recoge el valor del objeto y añade un paso con el atributo y el valor si tiene. Se añade al objeto el id del paso creado, el paso y el nodo. Por último, devuelve el nodo creado.

**public Node addPasoTerminalDes(String element, String atributo, Object object, Boolean haveBrother, Node nodeAnt):**

Recoge el valor del objeto y añade un paso con el atributo y el valor si tiene. Añade un nodo con el terminal y con el nodo anterior como padre. Se añade al objeto el paso y el nodo. Por último, devuelve el nodo creado.

**private String buildRule(Object[] objects):**

Por cada objeto, añade a la regla el nombre de la clase y si es un símbolo recursivo le añade un 1. Devuelve la regla.

**private Class getClass( Object ob):**

Busca la clase por el nombre.

**private Integer getId(Method id,Object object):**

Obtiene el id del objeto.

**private Method getMethod(Class clase, String methodName):**

Obtiene un método sin parámetro de una clase.

**private Method getMethodSetId(Class clase, String methodName):**

Obtiene el método setId del objeto.

**private Node getNode(Method node, Object object):**

Obtiene el nodo del objeto.

**private Paso getPaso(Method paso, Object object):**

Obtiene el paso del objeto.

**private String getValue(Method value, Object object)**

Obtiene el valor del objeto.

**private void setId(Method setId, Object object, Integer id):**

Añade el id al objeto.

**private void setNode(Class claseAnt, Object object, Node nodo):**

Añade el nodo al objeto.

**private void setPaso(Class claseAnt, Object object, Paso paso):**

Añade el paso al objeto.

**public Boolean writeXML():**

Escribe la parte de <contenido> y la parte de <arbol> del xml. Después crea el archivo XML.

**private String removeActions(String production):**

Dada una producción elimina las acciones y los símbolos recursivos.

Se separa la producción y por cada símbolo si no es una acción, si pertenece a los antecedentes se añade al resultado , si no pertenece se añade al resultado la parte del símbolo que termina cuando empiezan los dígitos.

**private String removeOnlyActions(String production):**

Dada una producción elimina las acciones.

Se separa la producción y por cada símbolo si no es una acción se añade al resultado.

**private Integer getNumberIndex(String symbol):**

Devuelve el índice donde empiezan los dígitos en un símbolo recursivo.

Por cada letra se suma uno hasta que se detecta un digito.

**private void readFile():**

Lee y guarda la gramática desde un archivo de texto.

Por cada línea la separa en antecedente y producciones y se separa las producciones. Se añade el antecedente a la lista si no está y se guarda en el mapa de la gramática con acciones una lista nueva con el antecedente como clave. Se añaden cada producción a la lista anteriormente mencionada.

**private void writeTraductor():**

Escribe la parte de <traductor> y <cadena> del XML.

Se crea la etiqueta “traductor” y se añade a la etiqueta espec. Se crea la etiqueta “tipo” y se añade a traductor, después se añade el texto “Descendente” a tipo. Por cada antecedente , por cada producción se añade la regla. Cuando se han añadido todas las reglas se crea la etiqueta “cadena” y se añade a espec, por último, se añade el texto de la cadena de entrada a cadena.

**private void addRule(String antecedent ,String production,Element traductor):**

Dado un antecedente , una producción y la etiqueta a la que añadir la regla , añade una regla al xml.

Forma el id de la regla añadiendo a “R” el contador de las reglas. Añade al mapa de ruleIds la regla con el id de la regla como clave. Crea la etiqueta “regla” y la añade a la etiqueta traductor, después le añade un atributo “id” con el id y recoge las acciones semánticas de la regla. Por cada acción semántica:

crea una etiqueta “accionSemantica”, la añade a la etiqueta de regla, obtiene la posición de la acción semántica a la regla para crear un atributo ”pos” con el valor obtenido. Añade como texto la acción semántica y si la acción está en el medio de la regla se añade una etiqueta “intermedio” con el texto sí.

Añade los símbolos de la producción. Y suma uno al contador de reglas.

**private ArrayList<String> actions(String production):**

Dada una producción devuelve todas las acciones en ella.

Separa la producción en símbolos y por cada símbolo si es una acción la añade a la lista que va a devolver.

**private Integer getPos(String action, String production):**

Dada una acción y una producción devuelve la posición detrás de la que se encuentra la acción.

Divide la producción en símbolos y mientras no sean igual a la acción si el símbolo no es una acción se suma uno a la posición y se pasa al siguiente símbolo.

**private void addSymbols(String production, Element regla,String antecedente):**

Dada una producción, un antecedente y la etiqueta regla escribe los símbolos de la producción en el xml.

Separa la producción en símbolos, por cada símbolo:

Si es el primero crea la etiqueta “simbolo” , la añade a regla , crea la etiqueta “valor” y la añade a símbolo. Si es la primera producción de la regla le añade a valor el antecedente si no añade “|”. Por último, crea la etiqueta “terminal”, la añade a símbolo y le añade el texto “false”.

Si el símbolo no es una acción crea la etiqueta “simbolo” , la añade a regla , crea la etiqueta “valor” y la añade a símbolo. Le añade a valor el símbolo. Crea la etiqueta “terminal”, la añade a símbolo y le añade el texto “false” si la primera letra es mayúscula y “true” si no.

private void writeArbol():

Escribe la parte de <arbol> del xml.

Actualiza el nivel de los nodos y recoge la altura del árbol. Crea la etiqueta árbol y la añade al documento. Crea la etiqueta numNodos y la añade la etiqueta arbol con el número de nodos como texto. Crea la etiqueta altura y la añade la etiqueta arbol con la altura del árbol como texto. Por cada nodo en la lista de nodos:

Creo la etiqueta nodo y la añado a la etiqueta arbol con un atributo con el id del nodo. Creo la etiqueta elemento y la añado a la etiqueta nodo con el elemento del nodo como texto, creo la etiqueta nivel y la añado a la etiqueta nodo con el nivel del nodo como texto, creo la etiqueta terminal y la añado a la etiqueta nodo con true si es un terminal y false si no como texto.

**private void writeContenido():**

Escribe la parte de <contenido> del xml.

Crea la etiqueta contenido y la añade al documento. Por cada paso de la lista pasos:

Crea la etiqueta paso y la añade a contenido con un atributo id. Crea la etiqueta tipo y la añade a contenido con el texto del tipo de paso, Si el paso tiene regla:

Crea la etiqueta nuevaRegla y la añade a paso con un atributo refRegla con el id de la regla y la regla como texto. Crea la etiqueta widthRegla y la añade a paso con el ancho de la regla como texto.

Crea la etiqueta cadena y la añade a paso. Crea la etiqueta leído y la añade a cadena con la cadena leído como texto. Crea la etiqueta pendiente y la añade a cadena con la cadena pendiente como texto. Crea la etiqueta elemento y la añade a paso con el elemento del paso como texto. Si tiene nodos relacionados crea la etiqueta relNodos y la añade a paso con los ids de los nodos relacionados del paso como texto. Crea la etiqueta valor y la añade a paso con el valor del paso como texto. Si tiene nodos que cambien de valor por él:

Crea la etiqueta accionSemanticaEjecutada y la añade a paso. Por cada nodo que cambia de valor:

Crea la etiqueta nodo y la añade a accionSemanticaEjecutada. Crea la etiqueta refNodo y la añade a nodo con el id del nodo que cambia. Crea la etiqueta atributos y la añade a nodo con el valor al que cambia el nodo.

**private String writeRelNodes(HashSet<Integer> relNodo):**

Convierte un conjunto de ids de nodos en una String.

**private void readChain(String entryChainPath):**

Lee la cadena de entrada de un archivo.

Por cada línea del archivo por cada elemento de la cadena (tiene que estar separados por un espacio), lo añade a la pila de cadena pendiente.

**private String writePendChain():**

Transforma la pila de cadena pendiente en un String.

**private Integer updateNode():**

Actualiza el nivel de los nodos.

Recoge la raíz del árbol. Por cada nodo:

Calcula la distancia desde el nodo a la raíz, actualiza el nodo, se queda con el nivel más alto.

Devuelve la altura del árbol sintáctico.

**private Integer distanciaARaiz(Paso raiz, Node objetivo):**

Cuento los nodos padres hasta llegar a la raíz. Devuelve la cantidad de nodos padre.

**public Paso getStep(Integer id):**

Devuelve el paso que corresponda con el id dado.

**public void updatesValues(Paso paso,Node node, String value):**

Recoge y asigna los valores y los nodos que se van a actualizar a este paso. Recoge el padre del nodo introducido y mientras el nodo no tenga hermano derecho añade a la lista de ids de nodos que cambian el id del nodo actual y a la lista de valores el valor cambiado. Para finalizar el bucle cambia el nodo actual por el padre. Para terminar, añade a las actualizaciones el id del nodo que si tiene hermano y el valor actualizado de este valor.

**public void updateNoTerminals(String regla, String valor, Object antecedente, Object primerSimbolo)**

Actualiza la regla del primer no terminal de la producción y añade el valor al antecedente.

## Clase Annotator V0.1

Atributos:

Constructor:

**public Annotator(String grammar, String cadena):**

Si la ruta de la gramática termina en .cup es un ascendente que usa CUP por lo que manipula el archivo como un CUP. Si no como un ANTLR.

Métodos:

**private String addNameToSymbols(String[] symbols):**

Por cada símbolo añade un nombre para referirse a él en cada regla.

**private void createClass(String s):**

Crea una clase para el símbolo con los atributos para un paso, un nodo, un valor String y un id. Además, añade su constructor y los gets y sets de los atributos.

**private String getNoTerminals(String line):**

Obtiene el no terminal de la línea.

**private HashSet<String> getTerminals(String line):**

Obtiene el terminal de la línea.

**private String newNoTerminals(HashSet<String> terminals):**

Añade los no terminales correspondientes a los terminales en la cabecera del CUP.

**private String noTerminalsOfTerminals(HashSet<String> terminals):**

Añade las reglas de los terminales necesarias para recoger los pasos en los que se procesan los terminales en el cup.

**private void writeAntlr(String grammar, String cadena):**

Edita el archivo .g4 correspondiente a ANTLR. Crea un nuevo archivo .g4. Por cada línea del archivo original si no es una línea vacía:

Si es una línea donde comienza el @header se pasa a true el booleano correspondiente y se escribe los import de node, paso y writer.

Si es una línea con el primer no terminal y no había imports se añaden y se añade el espacio que crea el writer.

Si no es una línea vacía y es un terminal se añade al String de nuevos terminales la regla de no terminales correspondiente y se crea la clase correspondiente al terminal.

Si no es una línea vacía y es un no terminal se crea la clase con el no terminal

Si no se ha llegado a la parte de los terminales en el archivo se escribe la line leída. Si no se añade la línea a un String.

Se escriben los nuevos no terminales y los terminales.

**private void writeCup(String grammar, String cadena):**

Escribe un archivo .cup nuevo editado para el correcto funcionamiento de la API a partir del original. Por cada línea si no está vacía:

Si es la línea del primer terminal escribe el espacio de variables globales inicializando el writer.

Si la línea es de declaración de terminales se guardan los terminales.

Si es la primera línea de no terminales se crean las clases de los nuevos no terminales y se escriben los nuevos no terminales creados a partir de los terminales recogidos.

Si la línea es de declaración de no terminales se guarda el no terminal y se crea una clase para él.

Si es una producción se les añaden los nombres a los símbolos del consecuente.

Si es un import se añaden los import de Paso, Node y Writer.

Se escribe la línea.

Por ultimo se escribe todas las reglas de los no terminales nuevos creados a partir de los terminales.

# VisTDSApiXMLCreator V0.6

## Clase VisTDSApiXMLCreator V0.2

Recoge como argumentos la gramática .cup o .g4, el archivo con la cadena de entrada y el directorio de destino de las clases que se van a crear. Si la gramática es un .cup llama al anotador para que cambie la gramática, después llama a CUP para que produzca las clases necesarias de CUP y, por último, mueve esas clases al destino que se haya introducido. Si es un .g4 llama al anotador, después llama a ANTLR para que produzca las clases necesarias de ANTLR y mueve dichas clases al destino dado.

**Métodos:**

**private static void moveFilesANTLR(String nombreArchivo, String destino):**

Si existen se mueven todos los archivos producidos por ANTLR al destino.

**private static void moveFilesCup(String destino):**

Si existen se mueven todos los archivos producidos por CUP al destino.

## Clase Node V0.3

Representa los nodos de la parte de árbol del XML

**Atributos:**

**private Integer id:** id del nodo, se crea automáticamente según vas añadiendo nodos.

**private String element:** elemento del nodo.

**private Boolean terminal:** true si es terminal false si no lo es.

**private Integer nivel:** niveldel árbol en el que se encuentra.

**private Node fahterNode:** nodo padre en el árbol sintáctico.

**private Boolean haveBrother:** true si el nodo tiene hermano derecho y false si no. Este atributo esta pensado para usarse en los traductores descendentes para saber cuando parar de actualizar valores.

**Constructor:**

**public Node(Integer id, String element, Boolean terminal, Integer nivel** **, Boolean haveBrother):**

Recoge los datos pertenecientes a todos los atributos y los asigna.

## Clase Paso V0.2

**Atributos:**

**private Integer id:** representa el id del paso, el cual se asigna automáticamente cuando se añaden los pasos.

**private String tipo:** describe el tipo de paso(“DespDes” para desplazamientos en descendentes, “derivacion” para derivaciones, “desplazamiento” para desplazamientos en ascendentes y “reduccion” para reducciones ).

**private String leído:** parte de la cadena leída.

**private String pendiente:** parte dela cadena pendiente.

**private String elemento:** elemento que se está procesando.

**private String valor:** valor del elemento que se está procesando.

**private String regla:** regla correspondiente al paso (dicha regla solo se añade si es el primer elemento de la regla )

**private HashSet<Integer> relNodo:** nodo que produce este paso. (paso padre).

**private ArrayList<Integer> changedNodes:** nodos que cambian después de este paso.

**private ArrayList<String> changes:** valor de los pasos que van a cambiar después de este paso.

**Constructor:**

**public Paso(Integer id, String tipo, String leido, String pendiente, String elemento, String valor,** **HashSet<Integer>, String regla)**

Recoge los valores de los atributos e inicializa las listas **changedNodes** y **changes.**

## Clase Writer V0.6

**Atributos:**

**private HashMap<String, ArrayList<String>> grammar:** corresponde a la gramática , es un mapa que contiene las producciones por cada antecedente.

**private String path:** ruta de la gramática.

**private String pathResult:** ruta donde guardar el xml.

**private Stack<String> pendChain:** pila de String que representan la cadena que aún no se ha procesado.

**private String readChain:** String que representa la parte de la cadena leída.

**private Element espec:** corresponde al elemento a partir del que se va a escribir las partes del traductor , la cadena , el árbol y el contenido.

**private Document doc:** representa el documento XML.

**private Integer ruleCount:** contador de reglas.

**private ArrayList<String> antecedentes:** lista con los antecedentes de la gramática.

**private Integer numNodos:** contador de nodos.

**private ArrayList<Node> nodes:** lista de nodos.

**private ArrayList<Paso> steps:** lista de pasos.

**private HashMap<String, String> ruleId:** mapa con los ids de cada regla.

**private Integer pasoCount:** contador de pasos.

**private String traductorType:** representa el tipo de traductor(Ascendente o Descendente) **.**

**private HashMap<Integer,Paso>stepMaps:** mapa que guarda los pasos por id para poder recuperarlos de una forma rápida y así actualizar los valores del paso.

**Constructor:**

**public Writer( String path, String pathResult, String entryChainPath, Boolean isDescendat):**

Recoge la ruta de la gramática, la ruta de destino y la ruta de la cadena de entrada y si es descendente (true) o ascendente (false).

Inicializa los mapas de la gramática ,de los ids por regla y de los pasos por id y las listas de antecedentes, pasos y nodos. Lee la gramática y la guarda en los mapas. Inicializa la pila de cadena pendiente y lee la cadena del archivo. Si es descendente(true) de le asigna “Descendente” al tipo de traductor y si es ascendente(false) “Ascendente”. Inicializa el documento XML y escribe la parte de <traductor> del XML.

**Métodos:**

**public Node addNode(String element, Boolean terminal, Integer nivel** **,Boolean haveBrother):**

Crea un nodo, lo añade a la lista de nodos y aumenta el contador de nodos. Devuelve el nodo creado.

**public Paso addPaso(Boolean isDisplacement, String elementoLeido, String element, String valor, String regla,HashSet<Integer> relNodo):**

Recoge si un paso es un desplazamiento, el elemento de la cadena que se haya podido leer en ese paso, el elemento procesado en ese paso , el valor del paso , la regla que puede tener ese paso y los nodos con los que se relaciona.

Si se ha procesado un terminal se saca de la cadena de pendientes. Se convierte la cadena de pendientes a String.

Si el paso era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “despDes”. Crea un paso.

Si el paso era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “desplazamineto”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “derivacion”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “reduccion”. Crea un paso.

Añade a la lista de pasos y al mapa el paso creado y aumenta el contador de pasos.

Devuelve el paso creado.

**public Paso addPaso(Boolean isDisplacement,String elementoLeido,String element, String valor, String regla):**

Recoge si un paso es un desplazamiento, el elemento de la cadena que se haya podido leer en ese paso, el elemento procesado en ese paso , el valor del paso y la regla que puede tener ese paso. Se usa para nodos terminales en el traductor ascendente.

Si se ha procesado un terminal se saca de la cadena de pendientes. Se convierte la cadena de pendientes a String.

Si el paso era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “despDes”. Crea un paso.

Si el paso era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “desplazamineto”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “derivacion”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “reduccion”. Crea un paso.

Añade a la lista de pasos y al mapa el paso creado y aumenta el contador de pasos.

Devuelve el paso creado.

**public Paso addPasoPrimero( String element, String valor, String regla):**

Recoge el elemento procesado en ese paso , el valor del paso y la regla que puede tener ese paso. Se usa para el primer nodo en un traductor descendente.

Se convierte la cadena de pendientes a String.

Crea un paso. Con el tipo de paso “primero”.

Añade a la lista de pasos y al mapa el paso creado y aumenta el contador de pasos.

Devuelve el paso creado.

**public Paso addPaso(Boolean isDisplacement,String elementoLeido,String element, String valor, String regla,Integer relNodo):**

Recoge si un paso es un desplazamiento, el elemento de la cadena que se haya podido leer en ese paso, el elemento procesado en ese paso , el valor del paso, la regla que puede tener ese paso y el nodo con el que se relaciona. Se usa para nodos con un único nodo relacionado como en el caso de los traductores ascendentes.

Crea un conjunto de Integer y añade el nodo relacionado. Si se ha procesado un terminal se saca de la cadena de pendientes. Se convierte la cadena de pendientes a String.

Si el paso era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “despDes”. Crea un paso.

Si el paso era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “desplazamineto”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era descendente el tipo de paso será “derivacion”. Crea un paso.

Si el paso no era un desplazamiento y el tipo de traductor era ascendente el tipo de paso será “reduccion”. Crea un paso.

Añade a la lista de pasos y al mapa el paso creado y aumenta el contador de pasos.

Devuelve el paso creado.

**public Node addPasoLambda(String element, String atributo, Object object):**

Crea una regla con el no terminal y lambda. Añade dos nodos, uno con lambda y otro con el no terminal. Recoge la clase del objeto y el método getValue. Añade dos pasos uno para lambda y otro para el no terminal. Añade el id, el nodo y el paso al no terminal. Por último, devuelve el nodo del no terminal.

**public Node addPasoLambdaDes(String element, String atributoHer, String atributoSint, Object object, Boolean haveBrother, String her, Node nodeAnt):**

Crea una regla con el no terminal y lambda. Recoge la clase del objeto. Añade un nodo con el no terminal (con el booleano de si tiene hermano) y un paso uno para el no terminal (con los atributos y her como valor). Añade al nuevo nodo el anterior como padre e introduce uno nuevo con lambda. Añade a este nodo el que hemos creado antes como padre y crea un paso con lambda. Añade el nodo y el paso al no terminal. Por último, actualiza los valores de los pasos conectados y devuelve el nodo del no terminal.

**public Node addPasoNoTerminal(String element, String atributo, String/Integer value, String rule, Object ... objects):**

Añade un nuevo nodo con el elemento. Por cada objeto empezando por el segundo recoge la clase, el id y el nodo, añade el id a nodos relacionados y el nodo como padre al que hemos creado. Si es el segundo objeto se le añade la regla al paso. Se añade el paso con el no terminal y el valor del primer objeto. Por último, se añade el id del paso creado, el nodo y el paso al primer objeto y devuelve el nodo del no terminal.

**public Node addPasoNoTerminalDes(String element, String atributoHer, String atributoSint, Object object, Boolean haveBrother, String her, Node nodeAnt):**

Recoge la clase del objeto. Si es el primer paso añade el nodo y el primer paso. Si no se añade el nodo con el elemento y el paso con un atributo o con dos si tiene atributo heredado. Si no es recursivo añade el nodo con el elemento y el paso con un atributo o con dos si tiene atributo heredado. Por último, se añade el nodo anterior como padre al creado y el nodo y el paso al objeto, además devuelve el nodo.

**public Node addPasoTerminal(String element, String atributo, Object object):**

Añade un nodo con el terminal. Recoge el valor del objeto y añade un paso con el atributo y el valor si tiene. Se añade al objeto el id del paso creado, el paso y el nodo. Por último, devuelve el nodo creado.

**public Node addPasoTerminalDes(String element, String atributo, Object object, Boolean haveBrother, Node nodeAnt):**

Recoge el valor del objeto y añade un paso con el atributo y el valor si tiene. Añade un nodo con el terminal y con el nodo anterior como padre. Se añade al objeto el paso y el nodo. Por último, devuelve el nodo creado.

**private String buildRule(Object[] objects):**

Por cada objeto, añade a la regla el nombre de la clase y si es un símbolo recursivo le añade un 1. Devuelve la regla.

**private Class getClass( Object ob):**

Busca la clase por el nombre.

**private Integer getId(Method id,Object object):**

Obtiene el id del objeto.

**private Method getMethod(Class clase, String methodName):**

Obtiene un método sin parámetro de una clase.

**private Method getMethodSetId(Class clase, String methodName):**

Obtiene el método setId del objeto.

**private Node getNode(Method node, Object object):**

Obtiene el nodo del objeto.

**private Paso getPaso(Method paso, Object object):**

Obtiene el paso del objeto.

**private String getValue(Method value, Object object)**

Obtiene el valor del objeto.

**private void setId(Method setId, Object object, Integer id):**

Añade el id al objeto.

**private void setNode(Class claseAnt, Object object, Node nodo):**

Añade el nodo al objeto.

**private void setPaso(Class claseAnt, Object object, Paso paso):**

Añade el paso al objeto.

**public Boolean writeXML():**

Escribe la parte de <contenido> y la parte de <arbol> del xml. Después crea el archivo XML.

**private String removeActions(String production):**

Dada una producción elimina las acciones y los símbolos recursivos.

Se separa la producción y por cada símbolo si no es una acción, si pertenece a los antecedentes se añade al resultado , si no pertenece se añade al resultado la parte del símbolo que termina cuando empiezan los dígitos.

**private String removeOnlyActions(String production):**

Dada una producción elimina las acciones.

Se separa la producción y por cada símbolo si no es una acción se añade al resultado.

**private Integer getNumberIndex(String symbol):**

Devuelve el índice donde empiezan los dígitos en un símbolo recursivo.

Por cada letra se suma uno hasta que se detecta un digito.

**private void readFile():**

Lee y guarda la gramática desde un archivo de texto.

Por cada línea la separa en antecedente y producciones y se separa las producciones. Se añade el antecedente a la lista si no está y se guarda en el mapa de la gramática con acciones una lista nueva con el antecedente como clave. Se añaden cada producción a la lista anteriormente mencionada.

**private void writeTraductor():**

Escribe la parte de <traductor> y <cadena> del XML.

Se crea la etiqueta “traductor” y se añade a la etiqueta espec. Se crea la etiqueta “tipo” y se añade a traductor, después se añade el texto “Descendente” a tipo. Por cada antecedente , por cada producción se añade la regla. Cuando se han añadido todas las reglas se crea la etiqueta “cadena” y se añade a espec, por último, se añade el texto de la cadena de entrada a cadena.

**private void addRule(String antecedent ,String production,Element traductor):**

Dado un antecedente , una producción y la etiqueta a la que añadir la regla , añade una regla al xml.

Forma el id de la regla añadiendo a “R” el contador de las reglas. Añade al mapa de ruleIds la regla con el id de la regla como clave. Crea la etiqueta “regla” y la añade a la etiqueta traductor, después le añade un atributo “id” con el id y recoge las acciones semánticas de la regla. Por cada acción semántica:

crea una etiqueta “accionSemantica”, la añade a la etiqueta de regla, obtiene la posición de la acción semántica a la regla para crear un atributo ”pos” con el valor obtenido. Añade como texto la acción semántica y si la acción está en el medio de la regla se añade una etiqueta “intermedio” con el texto sí.

Añade los símbolos de la producción. Y suma uno al contador de reglas.

**private ArrayList<String> actions(String production):**

Dada una producción devuelve todas las acciones en ella.

Separa la producción en símbolos y por cada símbolo si es una acción la añade a la lista que va a devolver.

**private Integer getPos(String action, String production):**

Dada una acción y una producción devuelve la posición detrás de la que se encuentra la acción.

Divide la producción en símbolos y mientras no sean igual a la acción si el símbolo no es una acción se suma uno a la posición y se pasa al siguiente símbolo.

**private void addSymbols(String production, Element regla,String antecedente):**

Dada una producción, un antecedente y la etiqueta regla escribe los símbolos de la producción en el xml.

Separa la producción en símbolos, por cada símbolo:

Si es el primero crea la etiqueta “simbolo” , la añade a regla , crea la etiqueta “valor” y la añade a símbolo. Si es la primera producción de la regla le añade a valor el antecedente si no añade “|”. Por último, crea la etiqueta “terminal”, la añade a símbolo y le añade el texto “false”.

Si el símbolo no es una acción crea la etiqueta “simbolo” , la añade a regla , crea la etiqueta “valor” y la añade a símbolo. Le añade a valor el símbolo. Crea la etiqueta “terminal”, la añade a símbolo y le añade el texto “false” si la primera letra es mayúscula y “true” si no.

private void writeArbol():

Escribe la parte de <arbol> del xml.

Actualiza el nivel de los nodos y recoge la altura del árbol. Crea la etiqueta árbol y la añade al documento. Crea la etiqueta numNodos y la añade la etiqueta arbol con el número de nodos como texto. Crea la etiqueta altura y la añade la etiqueta arbol con la altura del árbol como texto. Por cada nodo en la lista de nodos:

Creo la etiqueta nodo y la añado a la etiqueta arbol con un atributo con el id del nodo. Creo la etiqueta elemento y la añado a la etiqueta nodo con el elemento del nodo como texto, creo la etiqueta nivel y la añado a la etiqueta nodo con el nivel del nodo como texto, creo la etiqueta terminal y la añado a la etiqueta nodo con true si es un terminal y false si no como texto.

**private void writeContenido():**

Escribe la parte de <contenido> del xml.

Crea la etiqueta contenido y la añade al documento. Por cada paso de la lista pasos:

Crea la etiqueta paso y la añade a contenido con un atributo id. Crea la etiqueta tipo y la añade a contenido con el texto del tipo de paso, Si el paso tiene regla:

Crea la etiqueta nuevaRegla y la añade a paso con un atributo refRegla con el id de la regla y la regla como texto. Crea la etiqueta widthRegla y la añade a paso con el ancho de la regla como texto.

Crea la etiqueta cadena y la añade a paso. Crea la etiqueta leído y la añade a cadena con la cadena leído como texto. Crea la etiqueta pendiente y la añade a cadena con la cadena pendiente como texto. Crea la etiqueta elemento y la añade a paso con el elemento del paso como texto. Si tiene nodos relacionados crea la etiqueta relNodos y la añade a paso con los ids de los nodos relacionados del paso como texto. Crea la etiqueta valor y la añade a paso con el valor del paso como texto. Si tiene nodos que cambien de valor por él:

Crea la etiqueta accionSemanticaEjecutada y la añade a paso. Por cada nodo que cambia de valor:

Crea la etiqueta nodo y la añade a accionSemanticaEjecutada. Crea la etiqueta refNodo y la añade a nodo con el id del nodo que cambia. Crea la etiqueta atributos y la añade a nodo con el valor al que cambia el nodo.

**private String writeRelNodes(HashSet<Integer> relNodo):**

Convierte un conjunto de ids de nodos en una String.

**private void readChain(String entryChainPath):**

Lee la cadena de entrada de un archivo.

Por cada línea del archivo por cada elemento de la cadena (tiene que estar separados por un espacio), lo añade a la pila de cadena pendiente.

**private String writePendChain():**

Transforma la pila de cadena pendiente en un String.

**private Integer updateNode():**

Actualiza el nivel de los nodos.

Recoge la raíz del árbol. Por cada nodo:

Calcula la distancia desde el nodo a la raíz, actualiza el nodo, se queda con el nivel más alto.

Devuelve la altura del árbol sintáctico.

**private Integer distanciaARaiz(Paso raiz, Node objetivo):**

Cuento los nodos padres hasta llegar a la raíz. Devuelve la cantidad de nodos padre.

**public Paso getStep(Integer id):**

Devuelve el paso que corresponda con el id dado.

**public void updatesValues(Paso paso,Node node, String value):**

Recoge y asigna los valores y los nodos que se van a actualizar a este paso. Recoge el padre del nodo introducido y mientras el nodo no tenga hermano derecho añade a la lista de ids de nodos que cambian el id del nodo actual y a la lista de valores el valor cambiado. Para finalizar el bucle cambia el nodo actual por el padre. Para terminar, añade a las actualizaciones el id del nodo que si tiene hermano y el valor actualizado de este valor.

**public void updateNoTerminals(String regla, String/Integer valor, Object antecedente, Object primerSimbolo)**

Actualiza la regla del primer no terminal de la producción y añade el valor al antecedente.

## Clase Annotator V0.2

Atributos:

**private HashSet<String> terminalsWithValue:** contiene todos los terminales de los que el usuario pide el valor.

Constructor:

**public Annotator(String grammar, String cadena** **, String destino):**

Si la ruta de la gramática termina en .cup es un ascendente que usa CUP por lo que manipula el archivo como un CUP. Si no como un ANTLR.

Métodos:

**private String addNameToSymbols(String[] symbols):**

Por cada símbolo añade un nombre para referirse a él en cada regla en el caso de que el usuario no lo haya añadido.

**private void createClass(String s):**

Crea una clase para el símbolo con los atributos para un paso, un nodo, un valor String y un id. Además, añade su constructor y los gets y sets de los atributos.

**private String getNoTerminals(String line):**

Obtiene el no terminal de la línea y lo añade al conjunto de terminales con valor si el usuario lo ha pedido.

**private HashSet<String> getTerminals(String line):**

Obtiene el terminal de la línea.

**private String newNoTerminals(HashSet<String> terminals):**

Añade los no terminales correspondientes a los terminales en la cabecera del CUP.

**private String noTerminalsOfTerminals(HashSet<String> terminals):**

Añade las reglas de los terminales necesarias para recoger los pasos en los que se procesan los terminales en el cup.

**private void writeAntlr(String grammar, String cadena):**

Edita el archivo .g4 correspondiente a ANTLR. Crea un nuevo archivo .g4. Por cada línea del archivo original si no es una línea vacía:

Si es una línea donde comienza el @header se pasa a true el booleano correspondiente y se escribe los import de node, paso y writer.

Si es una línea con el primer no terminal y no había imports se añaden y se añade el espacio que crea el writer.

Si no es una línea vacía y es un terminal se añade al String de nuevos terminales la regla de no terminales correspondiente y se crea la clase correspondiente al terminal.

Si no es una línea vacía y es un no terminal se crea la clase con el no terminal

Si no se ha llegado a la parte de los terminales en el archivo se escribe la line leída. Si no se añade la línea a un String.

Se escriben los nuevos no terminales y los terminales.

**private void writeCup(String grammar, String cadena):**

Escribe un archivo .cup nuevo editado para el correcto funcionamiento de la API a partir del original. Por cada línea si no está vacía:

Si es la línea del primer terminal escribe el espacio de variables globales inicializando el writer.

Si la línea es de declaración de terminales se guardan los terminales.

Si es la primera línea de no terminales se crean las clases de los nuevos no terminales y se escriben los nuevos no terminales creados a partir de los terminales recogidos.

Si la línea es de declaración de no terminales se guarda el no terminal y se crea una clase para él.

Si es una producción se les añaden los nombres a los símbolos del consecuente.

Si es un import se añaden los import de Paso, Node y Writer.

Se escribe la línea.

Por último, se escribe todas las reglas de los no terminales nuevos creados a partir de los terminales.