PROYECTO FIN DE CARRERA

20 DE FEBRERO DE 2.009

GENERACIÓN AUTOMÁTICA DE PATRONES

RAÚL MATALLANOS MORENA- 100042068
INGENIERÍA INFORMÁTICA

TUTOR: MÓNICA MARRERO LLINARES
DEPARTAMENTO DE INFORMÁTICA

TUTOR: JORGE MORATO LARA
DIRECTOR

ESTA PÁGINA HA SIDO DEJADA EN BLANCO INTENCIONADAMENTE

Índice

Índice de Tablas

Índice de Ilustraciones

1 OBJETIVO

Las herramientas de etiquetado semántico relacionan un recurso, como por ejemplo una página Web, o parte de ella, con un identificador, generalmente una URI (Universal Resource Indentifiers). Una URI determina inequívocamente un concepto que se menciona en el contenido del recurso. Estas anotaciones son la base del procesamiento automático de las páginas Web, por lo que la anotación semántica tiene una importancia crítica para conseguir que la Web Semántica llegue a ser una realidad. La Web Semántica implica que la información que en la versión actual de la Web es comprensible solamente por los seres humanos también esté disponible de una manera formal para sistemas inteligentes. Si el objetivo de la Web Semántica se cumple, serían concebibles nuevas aplicaciones para la Web, como pueden ser buscadores semánticos o agentes inteligentes[WEB1].

El objetivo del presente proyecto es el de desarrollar una herramienta visual de etiquetado semántico semiautomático. Para ello, se almacenarán los patrones, descritos como gramáticas, en ontologías OWL. Estos patrones se agruparán en clases dentro de las ontologías, y la herramienta las recuperará para localizar las entidades reconocidas por cada patrón presente en un texto suministrado por el usuario. El usuario podrá crear tantas clases como desee dentro de una ontología cargada en la herramienta.

La herramienta podrá crear un patrón nuevo en la ontología. Para ello, el usuario podrá crearlos de dos maneras distintas:

- Generarlos como gramáticas: el usuario especificará en la herramienta una gramática que modele el patrón, especificando todas las características de éste que desee.
- Generarlo a partir de una serie de ejemplos presentes en el texto cargado. La herramienta creará una gramática de manera automática que modele el comportamiento de un patrón que reconozca las entidades marcadas por el usuario y las similares.

Ambos tipos de patrones serán almacenados en la clase que el usuario elija dentro de la ontología cargada. Posteriormente, se podrán visualizar las gramáticas que modelan cada patrón, así como sus características, de manera que el usuario podrá modificarlo para ajustarlo a sus necesidades. De la misma manera, la herramienta posibilita al usuario la eliminación total de un patrón presente en la

ontología, y el almacenamiento permanente del patrón asociado a la clase de esa ontología.

El usuario podrá lanzar sobre el texto cargado uno o más patrones de la ontología seleccionada. Las entidades reconocidas por dichos patrones se marcarán sobre el texto. Para ello, se asignará un color a cada patrón, resaltándose cada entidad reconocida por cada patrón del color correspondiente.

La herramienta almacenará en la ontología, junto con los patrones, las entidades reconocidas. Para ello, tras realizar una búsqueda sobre el texto, el usuario podrá seleccionar aquellas entidades que desee que se almacenen en la ontología. Estas entidades podrán ser consultadas en cualquier momento a través del visor de ontologías presente en la herramienta.

A través del citado visor de ontologías, el usuario podrá eliminar cualquiera de las clases presentes en una ontología, eliminándose automáticamente todos los patrones y entidades relacionados con dicha clase Dicho visor persigue aumentar la usabilidad de la herramienta, facilitando la visualización y edición de patrones y entidades relacionadas con una clase, así como la visualización y edición de las relaciones jerárquicas de una ontología.

2 ESTADO DEL ARTE

Una de las características que dotaban al presente proyecto de mayor dificultad y a su vez lo hacía mucho más atractivo, era su temática. Aborda una rama de la extracción de información sobre la que actualmente se está trabajando mucho. Sin embargo, existen pocas referencias sobre herramientas similares a la que se pretende desarrollar. De hecho, no se he encontrado ninguna que realice exactamente lo que en este proyecto se expone. Este aspecto, lejos de ser un contratiempo, hace al proyecto mucho más atractivo e interesante. Existen varias herramientas que realizan acciones que tienen partes comunes con la herramienta que se pretende desarrollar y entre ellas encontramos como ejemplo de cada tipo las siguientes:

- Gate
- Ontotext Gazzetter
- KIM
- MnM

A continuación se hará una breve descripción de cada una de ellas.

GATE

Gate[GAT1] es una de las herramientas de procesamiento del lenguaje natural y recuperación de información más conocida. Se trata de un framework desarrollado en la Universidad de Sheffield en 1995 y ha sido utilizada en numerosos proyectos de investigación y desarrollo. La versión 1.0 de GATE fue lanzada en 1996, y utilizada en una amplia gama de contextos de análisis de lenguaje en diversos idiomas: Inglés, Griego, Español, Sueco, Alemán, Italiano y Francés. Actualmente se utiliza la versión 5 de la herramienta.

Para realizar tareas de reconocimiento de entidades, Annie[ANN1] (A Neraly-New Information Extraction System), el sistema proporcionado por Gate para tal efecto, utiliza listados, llamados "*Gazetteer Lists*". Si una entidad presente en el texto encaja con una de las palabras incluidas en el listado, la herramienta la etiqueta. Para ello, la palabra presente en el texto debe encajar totalmente (y no parte de ella) con la incluida en el listado. Es decir, debe estar entre espacios en blanco o signos de puntuación.

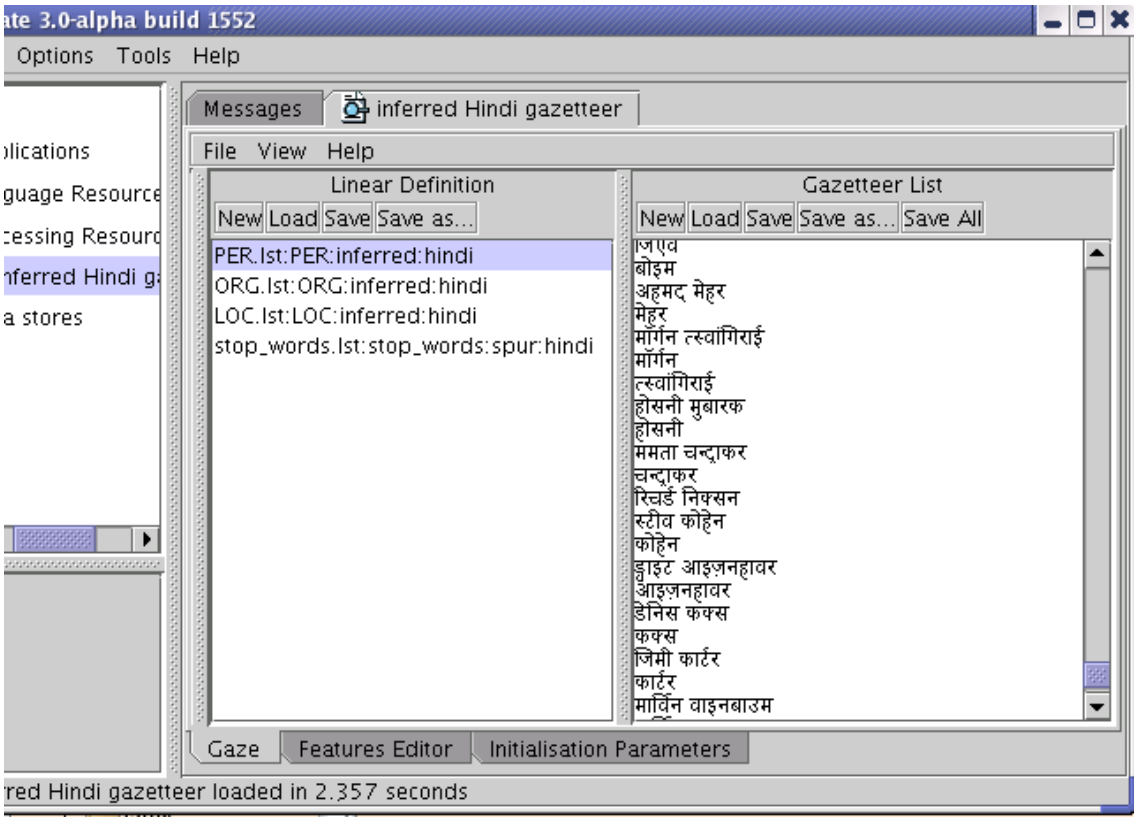


Ilustración 1: Gazetter List en Annie

Annie también permite reconocer entidades presentes en el texto descritas por una expresión regular. Para describir estas expresiones se utiliza un lenguaje propio

llamado Jape. Para ello, se le debe proporcionar la expresión regular en un conjunto de archivos, los cuales en conjunto componen el total de los autómatas de estados finitos que debe crear Gate para el reconocimiento de expresiones. Estos archivos deben respetar la sintaxis de Jape. GATE provee una herramienta denominada "Jape Debugger" la cual permite encontrar errores en programas JAPE.

Una gramática Jape consiste en un conjunto de frases, cada una de las cuales consiste en una serie de reglas patrón/acción. Estas fases se ejecutan secuencialmente, generando una cascada de reglas cuya finalidad es la de reconocer si una entidad implementa una expresión regular.

La parte izquierda de cada regla (LHS, left-hand-side) consiste en un patrón de anotación, el cual puede contener operadores de las expresiones regulares, como pueden ser *, ? ó +.

La parte derecha de la regla (RHS, right-hand-side) consiste en sentencias de manipulación de anotaciones. Anotaciones que se ajustan a la parte izquierda de una regla pueden ser referenciadas en la parte derecha por medio de etiquetas que se adjuntan a los patrones. Por ejemplo, una única regla es suficiente para identificar direcciones IP:

```
Rule: IPAddress (
{Token.kind == number}
{Token.string == "."}
{Token.kind == number}
{Token.string == "."}
{Token.kind == number}
{Token.string == "."}
{Token.kind == number} ) :ipAddress -->
:ipAddress.Address = {kind = "ipAddress"}
```

El contexto es utilizado para definir bajo qué situaciones se aplica un patrón. Por ejemplo, en el reconocimiento de una fecha, puede definirse un contexto en el que la fecha se reconozca solo si el año es precedido por las palabras "in" o "by". El contexto está limitado a la palabra justamente anterior a la entidad.

La parte derecha de una regla en JAPE puede contener código en lenguaje Java.

Esto es útil para remover anotaciones temporales y para manipular características de anotaciones previas.

En versiones anteriores, JAPE no era demasiado potente, pero se ha mejorado ostensiblemente incorporando el plug-in "*Montreal Transducer*" (Junio 2008), que hace posible la definición de patrones más flexibles:

- Los operadores *. + y ? son más restrictivos.
- El operador =~ en JAPE estándar busca anotaciones de expresiones regulares en cualquier lugar del valor, mientras que con "*Montreal Transducer*" ha de encajar la cadena de caracteres completa.
- Nuevos operadores, tales como {Token.length < 5}, {Lookup.minorType != "ignore"}, etc

A partir de la versión 3.1 de Gate, la herramienta añade el uso de ontologías. Para ello, proporciona una API formada por el paquete gate.creole.ontology y sus subpaquetes. La API proporciona un conjunto de operativas para trabajar con ontologías, pero no ofrece pleno apoyo a todas las operaciones que son posibles utilizando diferentes lenguajes de ontologías, como RDF-S, OWL o DAML. Sólo trata de proporcionar apoyo a todas las operaciones que todos estos formalismos tienen en común. Esto incluye las jerarquías de clases, las instancias y las propiedades. La terminología utilizada en la denominación de las clases y las operaciones está ligeramente orientada hacia RDF-S y OWL.

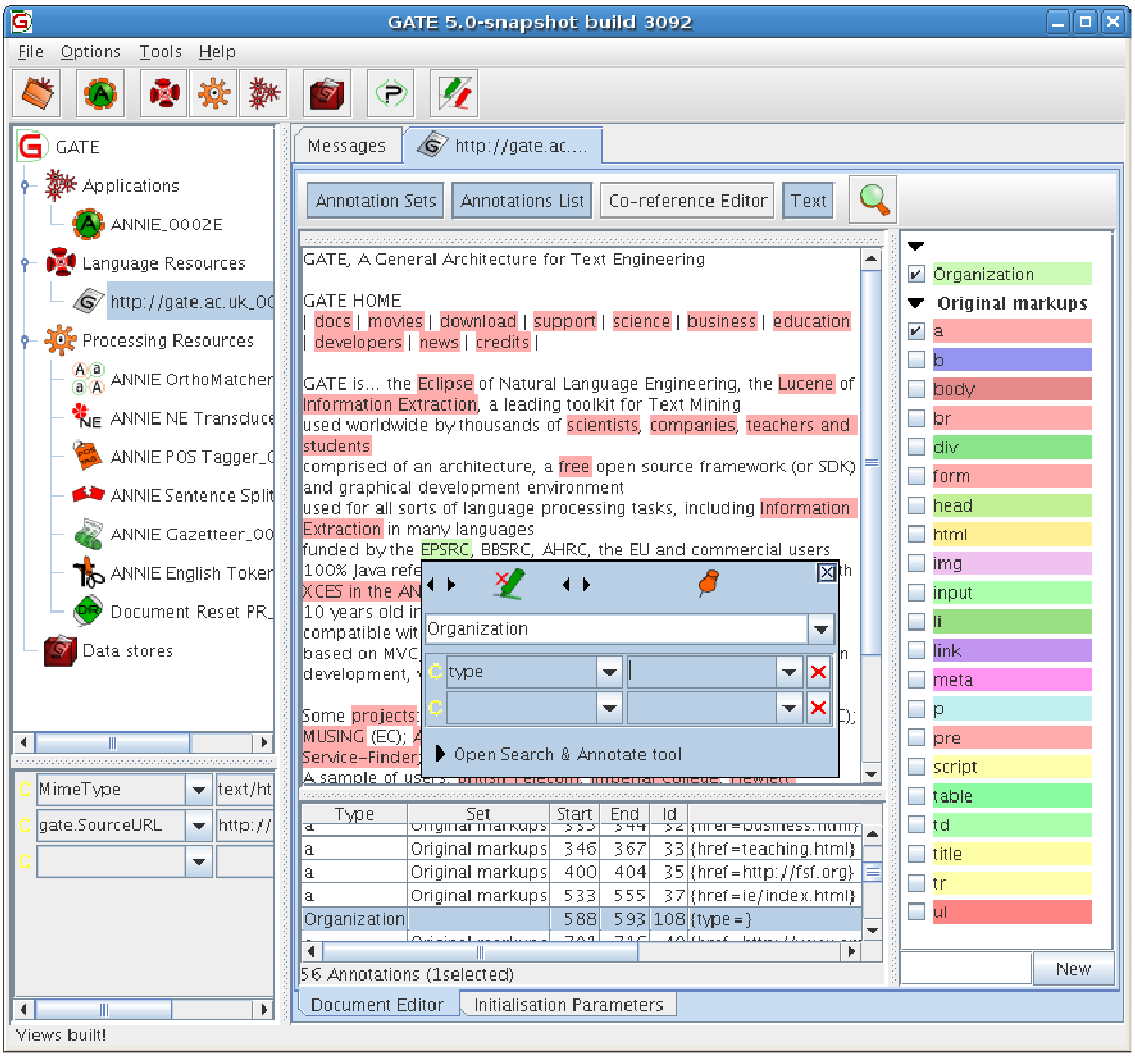


Ilustración 2: GUI de Gate

ONTOTEXT GAZZETTEER

OntoText Gazzetteer[ONT1] es una herramienta de etiquetado natural implementado por OntoText Lab[ONT2].

OntoText Gazzetteer es una interfaz que hace que las ontologías sean visibles en GATE soportando métodos básicos para el manejo de jerarquías. Para que OntoGazetteer funcione adecuadamente se debe contar con uno o más archivos de lista y dos archivos de definición. Los archivos de lista contienen una instancia por línea, por ejemplo, el archivo personas.lst contendría en cada línea el nombre de una persona. El primer archivo de definición usualmente denominado mappings.def describe las relaciones entre los archivos de lista y los conceptos ontológicos. El segundo archivo de definición contiene las relaciones entre los archivos de lista y las anotaciones que OntoGazetteer va a generar.

Cuando OntoGazetteer es ejecutado se generan anotaciones para todas las instancias existentes en los archivos de lista. Todas estas instancias serán anotadas con el mismo tipo de anotación denominado "Lookup" (del conjunto de anotaciones por defecto). Todas las anotaciones de tipo Lookup tendrán una característica que las diferenciaría según el archivo de lista al que pertenecen.

Esto no es muy útil ya que todos los conceptos de la ontología tendrían el mismo tipo de anotación indicado en el listado. Para transformarlas en anotaciones diferentes según su característica que las diferencia se utiliza el transductor JAPE descrito anteriormente.

A diferencia de este proyecto, con OntoGazetter no es posible generar anotaciones de modo automático o semiautomático, únicamente mediante listados. Además, el manejo de ontologías se antoja muy confuso y complicado, pues no se pueden visualizar de manera jerárquica.

KIM

La plataforma KIM [KIM1] [KIM2] permite la anotación semántica automática, indexación y recuperación de contenido no estructurado o semiestructurado. Los objetivos más importantes de Kim son:

- Generación de meta-datos para la Web Semántica, que permitan la hiper-vinculación y la visualización y navegación avanzadas
- Gestión del Conocimiento, mejorando la eficiencia de la actual indexación, recuperación, clasificación y filtrado de aplicaciones.

Para su funcionamiento, KIM analiza textos y reconoce las referencias a las entidades (como las personas, organizaciones, lugares, fechas). A continuación, se intenta hacer coincidir la referencia con una entidad conocida, con una descripción única y URI. Como alternativa, una nueva URI y descripción se generan automáticamente. Por último, la referencia en el documento es anotada con la URI de la entidad, pudiéndose consultar información de esa entidad recogida en otros documentos. Esta información es almacenada en ontologías, para su posterior recuperación.

Ilustración 3: GUI de KIM

Con el fin de permitir la facilidad de arranque de las aplicaciones, KIM está equipado con una ontología de nivel superior (PROTON) de alrededor de 250 clases y 100 propiedades, que puede ser modificada. Además, KIM incluye una base de conocimientos (KB KIM), pre-poblada con alrededor de 200.000 entidades.

Desde un punto de vista técnico, la arquitectura, basada en GATE, Sesame[SES1] y Lucene[LUC1], permite que las aplicaciones basadas en KIM puedan realizar anotaciones semánticas, recuperar el contenido de un texto basado en restricciones semánticas, así como la consulta y modificación de las ontologías y bases de conocimiento. La diferencia fundamental respecto al presente proyecto es que la detección de entidades se realiza de un modo automático, pero únicamente son reconocidas entidades de un conjunto de tipos predefinidos y con unas características específicas.

MnM

MnM[MNM1] es una herramienta de anotación desarrollada por "The Open University" que ofrece soporte automático y semiautomático para la anotación de páginas Web con contenido semántico[MNM2].

MnM se compone de un servidor de ontologías, y de un conjunto de herramientas de extracción de información, principalmente basadas en Procesamiento de Lenguaje Natural(NPL). Integra su propia ontología llamada Kimi. Utiliza el *wrapper induction*, con información lingüística. Los wrappers son los patrones de contenido semántico que un usuario da como ejemplo, de modo que el wrapper induction, consiste en el aprendizaje mediante inducción en base a patrones semánticos de ejemplo.

Sus herramientas lingüísticas soportan diversos idiomas, y es adaptable a muchos más. El *wrapper induction* combinado con entradas lingüísticas ofrece la posibilidad de trabajar sobre documentos no estructurados y HTML, y una interfaz amigable.

La herramienta está basada en Java y utiliza el API de Jena para el manejo de ontologías. Asimismo, presenta las ontologías de forma jerárquica, con sus clases e instancias. Permite también almacenar los resultados de una búsqueda en formato XML.

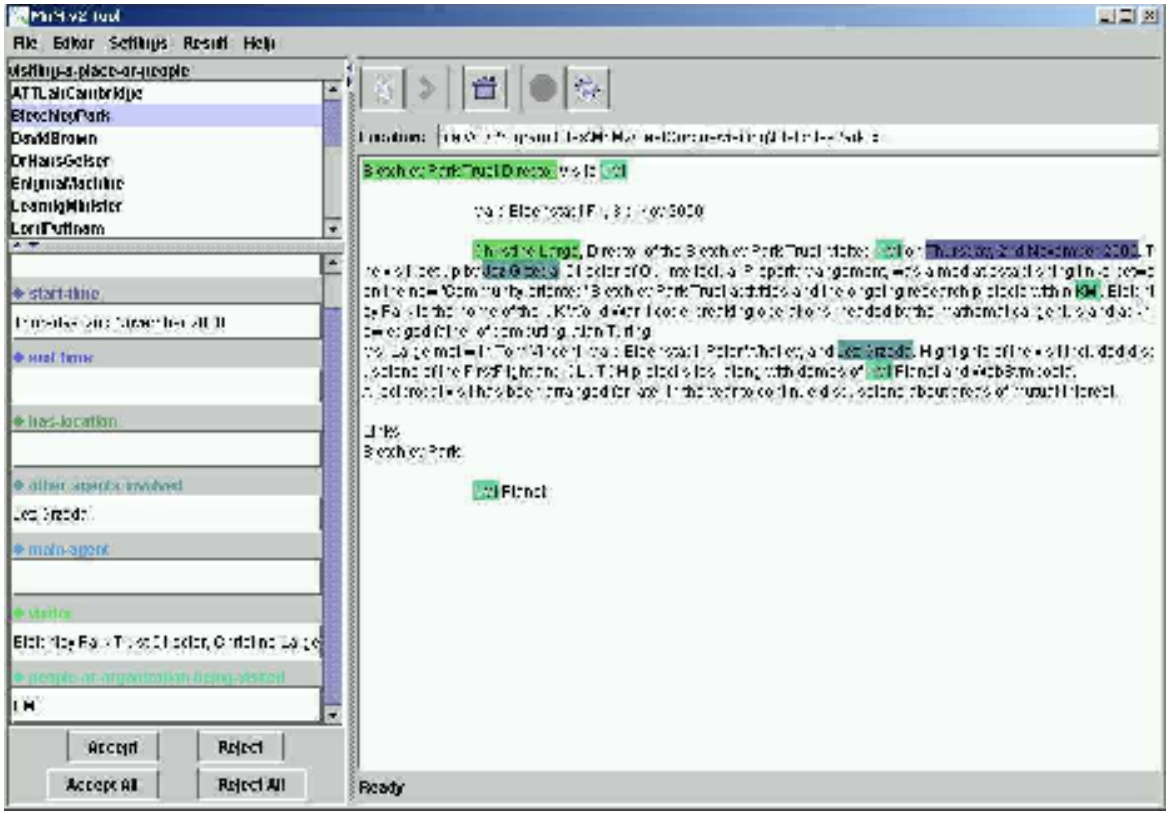


Ilustración 4: Interfaz MnM con marcado de entidades

Se trata pues, de una herramienta que comparte algunas de las características de la que aquí se pretende desarrollar; la diferencia fundamental radica en que si bien esta herramienta permite la anotación automática de cualquier tipo de entidades, lo realiza mediante aprendizaje automático sobre un corpus previamente anotado y generando modelos no editables para la posterior captura de entidades similares a las anotadas.

3 TECNOLOGÍA EMPLEADA

En este apartado se expone las características de las tecnologías utilizadas para la elaboración de la herramienta, así como la justificación de su uso.

OWL

OWL ("*Web Ontology Language*")[OWL1] es un lenguaje de Ontologías Web. Otros lenguajes anteriores utilizados para desarrollar herramientas y ontologías destinadas a comunidades específicas (especialmente para ciencias y aplicaciones específicas de comercio electrónico), no fueron definidos para ser compatibles con la arquitectura de la World Wide Web en general, y la Web Semántica en particular.

OWL rectifica esto proporcionando un lenguaje que utiliza la conexión proporcionada por RDF para añadir las siguientes capacidades a las ontologías:

- Capacidad de ser distribuidas a través de varios sistemas
- Escalable a las necesidades de la Web
- Compatible con los estándares Web de accesibilidad e internacionalización
- Abierto y extensible
- OWL extiende RDFS para permitir la expresión de relaciones complejas entre diferentes clases RDFS, y mayor precisión en las restricciones de clases y propiedades específicas. Esto incluye por ejemplo los recursos para limitar las propiedades de clases con respecto a número y tipo. Esta propiedad resulta indispensable para el almacenamiento de patrones en la ontología, para de esta forma poder definir las características exclusivas de cada patrón.

Dado que el presente proyecto sería extensible a la importación de ontologías desde la Web, OWL representa sin ningún lugar a dudas el lenguaje adecuado para almacenar las características de los patrones utilizados por la herramienta en ontologías. Además, se trata de un estándar rico y de uso extendido, por lo que, por extensión, haría la herramienta mucho más accesible.

A su vez OWL tiene tres variantes las cuales son *OWL-Lite*, *OWL-DL* y *OWL-Full*, diferenciándose entre ellas en la potencia ofrecida. Para las ontologías que hay que manejar en el presente proyecto, es suficiente con *OWL-Lite*, puesto que para

almacenar las características de los patrones sólo son necesarias propiedades muy básicas de OWL.

GRAMÁTICAS

Las gramáticas formales[GRAM1] [GRAM2] son sistemas de manipulación simbólica que permiten generar cadenas de símbolos, llamadas por esto gramáticas *bien formadas.* Es decir, una gramática es un ente formal para especificar, de una manera finita, el conjunto de cadenas de símbolos que constituyen un lenguaje

Una gramática es una cuádrupla:

$$G = (VT, VN, S, P)$$

Donde:

- VT= {conjunto finito de símbolos terminales}
- VN={conjunto finito de símbolos no terminales}
- S es el *símbolo inicial* y pertenece a VN
- P= {conjunto de producciones o de reglas de derivación}

Todas las cadenas del lenguaje definidas por la gramática están formadas con símbolos del *vocabulario terminal* VT. El vocabulario terminal se define por enumeración de los símbolos terminales. En el presente proyecto, el vocabulario terminal (VT), sería el conjunto de los caracteres del vocabulario castellano.

El *vocabulario no terminal* VN es el conjunto de símbolos introducidos como elementos auxiliares para la definición de la gramática, y que no figuran en las sentencias del lenguaje. Por convención, en el presente proyecto, un símbolo no terminal será un conjunto de caracteres en mayúsculas

La intersección entre el vocabulario terminal y no terminal es el conjunto vacío:

$$\{VN\} \cap \{VT\} = \{\emptyset\}$$

La unión entre el vocabulario terminal y no terminal es el *vocabulario*.

$$\{VN\} \cup \{VT\} = \{V\}$$

En ocasiones es importante distinguir si un determinado vocabulario incluye o no la cadena vacía, indicándose respectivamente con superíndice + o superíndice *, tal como se muestra a continuación:

$$V^{+} = V - \{\lambda\}$$

$$V^{*} = V + \{\lambda\}$$

En nuestro caso, en principio, las gramáticas reconocedoras de patrones no incluyen como tal el símbolo vacío.

El *símbolo inicial* S, o axioma de la gramática, es un símbolo no terminal a partir del cual se aplican las reglas de la gramática para obtener las distintas cadenas del lenguaje.

Las *producciones* P son las reglas que se aplican desde el símbolo inicial para obtener las cadenas del lenguaje. El conjunto de producciones P se define por medio de la enumeración de las distintas producciones, en forma de reglas.

Podemos ver a continuación dos ejemplos de gramáticas que reconocen cadenas de 1's separadas por 0's únicos

$$a)\quad \begin{aligned} S &\rightarrow 1S|1T|1 \\ T &\rightarrow 0S \end{aligned} \qquad b)\quad \begin{aligned} S &\rightarrow 0T|1S|1 \\ T &\rightarrow 0U|1S|1 \\ U &\rightarrow 0U|1U \end{aligned}$$

Vemos pues que un mismo lenguaje puede ser generado por varias gramáticas.

El lenguaje L (G) generado por una gramática G es el conjunto de todas las sentencias que puede generar G. Una sentencia pertenece a L (G) si:

- Está compuesta de símbolos terminales pertenecientes a la gramática (VT)
- La sentencia puede derivarse del símbolo inicial S aplicando las reglas de producción de la gramática

En nuestro caso, la primera de las restricciones se cumplirá siempre para textos en castellano, pues VT estará formada por todos los caracteres del alfabeto castellano. Por tanto, una sentencia, o conjunto de caracteres será reconocida por la gramática si puede derivarse del axioma utilizando las reglas de producción de la gramática

Existen varios tipos de gramáticas, según clasificó Chomsky[GRAM3]. A continuación se expone un breve resumen de ellas:

GRAMÁTICAS TIPO 0

Son las menos restrictivas. También se llaman *gramáticas no restringidas con* estructura *de frase.* La única restricción que presenta es:

- En la parte izquierda de cada regla tiene que haber al menos un símbolo no terminal.

Respecto a las partes derechas de las reglas, no hay ningún tipo de restricción, por lo que la única restricción es que no puede haber reglas de la forma:

A→ B donde A es la cadena vacía (λ).

GRAMÁTICAS TIPO 1

También llamadas *gramáticas sensibles al contexto.* Es decir que es importante tomar en cuenta la ubicación de los símbolos no terminales en la regla de derivación. Estos símbolos pueden preceder y/o suceder a símbolos terminales.

En este tipo de gramática sus reglas de producción son de la forma:

$$\alpha A \beta \rightarrow \alpha \gamma \beta$$

Estas gramáticas se llaman sensibles al contexto, pues únicamente se puede reemplazar A por γ siempre que estén en el contexto $\alpha \ldots \beta$. Es decir, para sustituir A por γ, A debe estar en el contexto establecido (en este caso precedido por α y a continuación β), y al sustituirse, el contexto ha de permanecer.

GRAMÁTICAS TIPO 2

Las Gramáticas de Tipo 2 también se denominan *gramáticas de contexto libre o libres de contexto.* Sus reglas de producción tan sólo admiten tener un símbolo no terminal en su parte izquierda, es decir son de la forma:

$$A \rightarrow \alpha$$

Donde $A \in VN$ y $\alpha \in (VN \cup VT)^+$

La denominación contexto libre se debe a que se puede sustituir A por a, independientemente del contexto en que aparezca A. Ya no es necesario que A aparezca en un contexto determinado para poder sustituirse por la parte derecha de la regla de producción. Para que una gramática sea considerada de Tipo 2, únicamente ha de cumplir que todas sus reglas estén compuestas en su parte izquierda por un único símbolo no terminal.

GRAMÁTICAS TIPO 3

También denominadas *regulares o gramáticas regulares a la derecha* comienzan sus reglas de producción por un símbolo terminal que puede ser seguido o no por un símbolo no terminal, es decir son de una de las formas siguientes:

A → aB

A→ a

Es decir, la parte derecha de la gramática siempre comenzará por un elemento no terminal. Estas gramáticas son las que generan lenguajes regulares (los obtenidos por medio de expresiones regulares).

Conviene resaltar, llegado este punto cuatro aspectos

- Para la definición de patrones reconocedores de entidades, se ha utilizado gramáticas de tipo 2. Es decir, por un lado son independientes del contexto, y por otro, las partes izquierdas de las reglas de producción están formadas por uno y solo un símbolo no terminal.
- Las reglas no derivarán en un conjunto de símbolos (terminales y no terminales), sino que derivarán en cinco formas posibles:
  - en un conjunto de no terminales
  - En conjunto de caracteres, de cualquier tipo (un conjunto de no terminales). Por ejemplo:

    A → "abd123"
  - En un rango numérico. Si por ejemplo tenemos la regla

    A → RangoNumerico{1,5}

    Equivaldría a:

    A → 1 | 2 | 3 | 4 | 5
  - En un rango alfabético. Si por ejemplo tenemos la regla

    A → RangoAlfabetico{a,e}

    Equivaldría a:

    A → a | b | c | d | e
  - En un rango Unicode. Para el presente proyecto, se ha tenido en cuenta los siguientes rangos:
    - Carácter
    - Letra
    - Puntuación de apertura
    - Puntuación de cierre
    - Otra puntuación
    - Separador
    - Otro símbolo

    Si por ejemplo tenemos la regla siguiente:

    A → RangoUnicode {Puntuación Apertura}

Equivaldría a:

A → **‘(‘** | **‘¿’** | **‘¡’** | **‘[‘** …….

- A la definición de gramáticas se ha añadido el término de la cardinalidad. Esto no es otra cosa que una simplificación de la manera de escribir gramáticas. Por tanto, una gramática a la que se añaden cardinalidades no es otra cosa que una gramática algo más compleja. Por ejemplo, si tenemos la siguiente gramática:

  A → B C

  B → "casa"

  C → {1,9}

  Y añadimos al no terminal "B" la cardinalidad mínima 1 y máxima 3, la gramática quedaría de la siguiente manera:

  A → B C | B B C | B B B C

  B → "casa"

  C → {1,9}

- Es necesario recordar que el hecho de que la gramática reconozca una cadena de caracteres no es condición sine qua non para que el patrón capture dicha entidad. Para que esto ocurra, la cadena de caracteres deberá cumplir otras restricciones, como de formato o contexto. Esta es una de las razones fundamentales por las que se optó por el uso de gramáticas en lugar de expresiones regulares, puesto que las primeras ofrecían mayor potencia para la adición de restricciones, tales como que dos símbolos de la gramática, y por ende dos partes de la cadena reconocida, tuvieran el mismo contenido o la misma cardinalidad. Además se escogió utilizar gramáticas de tipo 2, al proporcionarnos éstas mayor potencia en la generación de lenguajes que una expresión regular. Por ejemplo, nos permite representar un lenguaje en el que las palabras pertenecientes son aquellas que tienen un número determinado de A's seguido del mismo número de B's.

  S → ab | aAb

  A → ab | aAb

JENA

Para el almacenamiento de los patrones, como se ha citado anteriormente, se usarán ontologías con las que se relacionen. Se necesita pues, una tecnología capaz de cargar dichas ontologías, para posteriormente tratar la información obtenida de ellas en la herramienta. Asimismo, también tendría que ofrecer la posibilidad de manipular dichas ontologías, para poder albergar nueva información en ellas. El framework Jena ofrecía todas estas funcionalidades.

Jena[JEN1] es un Framework open source de java, creado por el equipo de Web Semántica de Hewlett-Packard, para la creación de aplicaciones para la Web Semántica. Jena se encuentra disponible como software de código abierto bajo licencia BSD (Berkeley System Distribution). Jena implementa las API para hacer gestionar RDF, RDFS, SPARQL y OWL.

Para los usuarios de Jena, la clase fundamental es "Model", una API creada para manejar las tripletas RDF. Un modelo puede ser creado desde un archivo local del sistema o desde un archivo remoto. Esta propiedad permitiría en un futuro importar las ontologías directamente desde la Web, sin necesidad de trabajar en local[JEN2].

Además, utilizando Jena podríamos almacenar la ontología en un soporte permanente. En el caso del recientemente aparecido SDB de Jena, por ejemplo, permite utilizar bases de datos basadas en Oracle, Ms SQLServer, DB2, PostgreSQL, MySQL, Apache Derby, H2 y HSQLDB. Cada almacén RDF se almacena en una base de datos independiente. Cada una de estas bases de datos contiene un conjunto de tablas orientadas a almacenar la información de las tripletas: Triples para las tripletas, Quads para los grafos RDF y Nodes para los nodos de las tripletas y grafos. SDB ofrece también diferentes layouts o definiciones de campos para estas tablas, cada uno con sus ventajas e inconvenientes. Una vez que la información está almacenada en el almacén es posible acceder a ella a través del API de Jena o mediante la ejecución de consultas SPARQL. Para nuestro proyecto no fue necesario albergar las ontologías en bases de datos (se almacenan en ficheros planos de texto), por lo que no se hizo uso de esta funcionalidad. De hecho, era suficiente con que nos brindara la funcionalidad necesaria para la lectura, manejo y escritura de las ontologías utilizadas en la aplicación.

Dado que Jena nos brinda las funcionalidades necesarias para realizar el proyecto, su uso es sencillo y está muy extendido, y está desarrollada en java, se decidió su uso frente a otras alternativas analizadas como SOFA[SOF1] o Protegé[PRO1] [PRO2].

JAVA

En este apartado se exponen las principales características del lenguaje de programación elegido para elaborar el presente proyecto. Pero la más influyente de todas fue haber escogido la posibilidad de utilizar la API de Jena para el tratamiento de ontologías, puesto que se trata de una API exclusiva de Java. Las características adicionales para haber realizado el proyecto en Java son las siguientes:

Simple

El único requerimiento para aprender Java es tener una comprensión de los conceptos básicos de la programación orientada a objetos.

Orientado a objetos

Java fue diseñado en este aspecto partiendo de cero, no es un derivado de otro lenguaje anterior y no tiene compatibilidad con ninguno de ellos. Java es un lenguaje orientado a objetos, eso implica que su concepción es muy próxima a la forma de pensar humana.

Potente

Pese a su simpleza se ha conseguido un considerable potencial, y aunque cada tarea se puede realizar de un número reducido de formas, se ha conseguido un gran potencial de expresión e innovación desde el punto de vista del programador.

Seguro

Es un lenguaje seguro: La máquina virtual, al ejecutar el código java, realiza comprobaciones de seguridad, además el propio lenguaje carece de características inseguras:

- Fuertes restricciones al acceso a memoria, como son la eliminación de punteros aritméticos y de operadores ilegales de transmisión.
- Rutina de verificación de bytecode que asegura que no se viole ninguna construcción del lenguaje
- Verificación del nombre de clase y de restricciones de acceso durante la carga.
- Sistema de seguridad de la interfaz que refuerza las medidas de seguridad en muchos niveles
- Además, en futuras versiones se prevé contar también con encriptación y técnicas similares.

El presente proyecto además, podría expandirse a la Web. Java ha sido diseñado poniendo un énfasis especial en el tema de la seguridad en la red, y se ha conseguido lograr cierta inmunidad en este aspecto, ya que un programa realizado en Java no puede realizar llamadas a funciones globales y acceder a recursos arbitrarios del sistema, por lo que el control sobre los programas ejecutables no es equiparable a otros lenguajes.

Robusto

Java verifica su código antes de ejecutarse, de manera que se consigue un alto margen de codificación sin errores. Se realiza un descubrimiento de la mayor parte de los errores durante el tiempo de compilación, ya que Java es estricto en cuanto a tipos y declaraciones, y lo que es rigidez y falta de flexibilidad se convierte en eficacia. Respecto a la gestión de memoria, Java libera al programador del compromiso de tener que controlar especialmente la asignación que de ésta hace a sus necesidades específicas. Este lenguaje posee una gestión avanzada de memoria llamada gestión de basura, y un manejo de excepciones orientado a objetos integrados. Estos elementos realizarán muchas tareas antes tediosas a la vez que obligadas para el programador. Ambas características son muy valoradas en este proyecto, debido a la gran cantidad de datos que pueden llegar a manejarse y al gran número de situaciones anómalas (excepciones) a tratar.

Interactivo

Uno de los requisitos de Java desde sus inicios fue la posibilidad de crear programas en red interactivos, por lo que es capaz de hacer varias cosas a la vez sin perder el rastro de lo que debería suceder y cuándo. Para que esto sea posible cuenta con múltiples hilos de utilización sencilla que le permiten pensar en el comportamiento específico que se intenta codificar, sin tener que integrarlo en un modelo de programación global de "bucle de suceso".

Dinámico

El lenguaje Java y su sistema de ejecución en tiempo real son dinámicos en la fase de enlazado. Las clases sólo se enlazan a medida que son necesitadas. Se pueden enlazar nuevos módulos de código bajo demanda, procedente de fuentes muy variadas, incluso desde la Red.

Multiplataforma

Java está diseñado para que un programa escrito en este lenguaje sea ejecutado correctamente independientemente de la plataforma en la que se esté actuando (Macintosh, PC, UNIX…). Para conseguir esto utiliza una compilación en una representación intermedia que recibe el nombre de código de byte de Java, que puede interpretarse en cualquier sistema con intérprete de Java. La desventaja de

un sistema de este tipo es el rendimiento; sin embargo, el hecho de que Java fuese diseñado para funcionar razonablemente bien en CPU de escasa potencia, unido a la sencillez de traducción a código máquina hacen que Java supere esa desventaja sin problemas.

Entorno de objeto rico

Existen varias clases que contienen las abstracciones básicas para el mundo con el que interaccionan sus programas. Se contará por tanto, con un conjunto de clases comunes que pueden crecer para admitir todas las necesidades del programador. Además la biblioteca de clases de Java proporciona un modelo unificador único de protocolos de Internet. Esta característica es fundamental para la elección de este lenguaje, pues se hace uso de librerías diseñadas para la generación de interfaces (Swing), o para la gestión de ontologías (Jena).

4 DESARROLLO

REQUISITOS

Para satisfacer los objetivos citados en el apartado anterior, lo primero que se debe realizar es identificar cuáles son los requisitos que debe implementar la herramienta a desarrollar.

Requisitos funcionales

A continuación se exponen los requisitos funcionales que ha de satisfacer la aplicación:

Identificador	RF-1
Nombre	Cargar texto
Descripción	Permitir al usuario cargar archivos de texto para la detección de patrones en su texto. Los formatos permitidos serán txt y rdf.

Identificador	RF-2
Nombre	Cargar ontología de usuario
Descripción	Permitir al usuario cargar ontologías (ontología de usuario).

Identificador	RF-3
Nombre	Visualizar ontología de usuario
Descripción	Posibilitar al usuario visualizar la ontología cargada en la herramienta. Se mostrará de forma jerárquica las clases e instancias almacenadas en dicha ontología de usuario.

Identificador	RF-4
Nombre	Visualizar patrones
Descripción	Permitir al usuario visualizarlos patrones relacionados con las clases de la ontología de usuario cargada previamente en el sistema.

Identificador	RF-5
Nombre	Visualizar patrones de una clase
Descripción	Permitir al usuario visualizar qué patrones almacenados en la ontología están relacionados con una determinada clase de la ontología de usuario seleccionada previamente.

Identificador	RF-6
Nombre	Consultar instancia
Descripción	El usuario podrá comprobar qué patrón recuperó una determinada instancia, previamente seleccionada y almacenada en la ontología de usuario.

Identificador	RF-7
Nombre	Crear clase
Descripción	Permitir crear una nueva clase en la ontología de usuario cargada previamente.

Identificador	RF-8
Nombre	Eliminar clase
Descripción	Permitir al usuario eliminar la clase seleccionada de la ontología de usuario cargada, eliminándose automáticamente: - las instancias relacionadas a dicha clase. - los patrones almacenados relacionados con esa clase. - sus subclases. - las instancias, los patrones y las subclases de las clases eliminadas con motivo del anterior punto, recursivamente.

Identificador	RF-9
Nombre	Crear patrón
Descripción	Permitir al usuario crear un patrón nuevo en una ontología de usuario cargada previamente en el sistema. El patrón creado pertenecerá a la clase que elija el usuario.

Identificador	RF-10
Nombre	Generar patrón
Descripción	Permitir al usuario generar un patrón, a partir de una serie de palabras seleccionadas sobre el texto cargado El patrón creado pertenecerá a la clase de la ontología de usuario seleccionada.

Identificador	RF-11
Nombre	Editar patrón
Descripción	Posibilitar al usuario editar un

Identificador	RF-12
Nombre	Eliminar patrón
Descripción	Posibilitar al usuario eliminar un patrón

Identificador	RF-13
Nombre	Buscar entidades
Descripción	Permitir al usuario lanzar sobre el texto cargado una búsqueda de las entidades presentes en el texto que encajen con uno de los patrones seleccionados, siendo visible para el usuario, para cada entidad, qué patrón la recuperó. La búsqueda es lanzada sobre todos los patrones seleccionados.

Identificador	RF-14
Nombre	Búsqueda única de entidades
Descripción	Si el usuario lanza la búsqueda de patrones seleccionando únicamente un patrón, se remarcarán en el visor de la ontología de usuario la clase a la que pertenece, así como las instancias capturadas por dicho patrón.

Identificador	RF-15
Nombre	Buscar siguiente entidad
Descripción	Visualizar los resultados de la búsqueda de entidades en el texto secuencialmente. La primera entidad visualizada será la primera en el texto, resaltada con el color asignado al patrón que la recuperó, y sucesivamente se irán visualizando las demás a medida que el usuario lo indique.

Identificador	RF-16
Nombre	Guardar entidades
Descripción	El usuario podrá seleccionar qué entidades de las localizadas en la búsqueda, quiere almacenar en la ontología de usuario cargada. Se almacenarán como instancias de la clase a la que pertenece el patrón que las recuperó.

Requisitos de los patrones

A continuación se exponen los requisitos identificados en la fase de análisis relativos a la estructura y organización de los patrones.

Identificador	RP-1
Nombre	Almacenamiento de los patrones
Descripción	Los patrones se almacenarán en una ontología (denominada ontología de patrones) OWL en un fichero de texto plano, en la misma ruta que la ontología de usuario. Esta ontología se servirá de una ontología única maestra, en la que se definen las propiedades de los patrones.

Identificador	RP-2
Nombre	Clase de los patrones
Descripción	Los patrones almacenarán la clase de la ontología de usuario a la cual pertenecen.

Identificador	RP-3
Nombre	Instancias de los patrones
Descripción	Los patrones almacenarán las instancias que hayan capturado en los textos (que el usuario haya decidido guardar) de manera que sea posible mantener una referencia de qué patrón ha capturado cada instancia.

Identificador	RP-4
Nombre	Prioridad de los patrones
Descripción	Los patrones almacenarán una prioridad, que determinará si captura una determinada entidad, en el caso de que ésta sea capturada por más patrones.

Identificador	RP-5
Nombre	Gramática de los patrones
Descripción	Los patrones almacenarán la gramática que determina la estructura de la entidad que han de reconocer.

Identificador	RP-6
Nombre	Terminales de la gramática

Descripción	Los nodos terminales de la gramática serán uno de los siguientes:
	- Literales: cadenas de texto invariables
	- Rango alfabético
	- Rango numérico
	- Rangos Unicode: uno de los siguientes
	  ○ Dígito
	  ○ Letra
	  ○ Puntuación de apertura
	  ○ Puntuación de cierre
	  ○ Otra puntuación
	  ○ Símbolo
	  ○ Separador

Identificador	RP-7
Nombre	Cardinalidad de los nodos no terminales
Descripción	Los patrones almacenarán la cardinalidad de los nodos no terminales de la gramática. Por defecto será de 1 como cardinalidad máxima y 1 como mínima.

Identificador	RP-8
Nombre	Nodos no terminales con igual cardinalidad
Descripción	Los patrones permitirán establecer que dos nodos no terminales de la gramática correspondiente han de tener la misma cardinalidad.

Identificador	RP-9
Nombre	Nodos no terminales con igual contenido
Descripción	Los patrones permitirán establecer que dos nodos no terminales de la gramática correspondiente han de tener el mismo contenido asignado (mismos caracteres de la entidad).

Identificador	RP-10
Nombre	Formato de los nodos no terminales
Descripción	Los patrones permitirán establecer para cada nodo no terminal, el formato que ha de tener el texto asignado, de entre uno de lo siguientes: - Mayúscula - Minúscula - Cursiva - Negrita - Subrayado

Identificador	RP-11
Nombre	Contexto del patrón
Descripción	Los patrones permitirán establecer una serie de palabras que han de estar en el contexto de la entidad (a una distancia establecida también por el usuario) para que ésta sea reconocida por el patrón.

Identificador	RP-12
Nombre	Excepciones del patrón
Descripción	Los patrones permitirán establecer, para cada nodo no terminal de la gramática, una excepción que consistirá en otro patrón. En caso de que una entidad reconocida lo sea también por el patrón establecido como excepción, haría que la entidad no fuera reconocida por el patrón.

Identificador	RP-13
Nombre	Formato de las excepciones
Descripción	Las excepciones estarán compuestas de otro patrón, pero simplificado. Solo constaran de gramáticas y cardinalidades, ignorándose las demás restricciones.

MODELO VISTA CONTROLADOR

Para el diseño del sistema, se opta por emplear la arquitectura MVC (Model - View - Controller), y más concretamente por su versión moderna, adaptada a la unificación de interfaces de entrada/salida en el GUI, más acorde con el tipo de aplicaciones que se desarrollan en la actualidad, donde las interfaces gráficas ocupan la práctica totalidad de la interacción entre los sistemas y los usuarios.

A continuación vamos a comentar brevemente cada uno de los módulos de la arquitectura:

- Modelo: incluye aquellas clases que se corresponden con el modelo de negocio, especificadas en el modelo conceptual del sistema y producto de las consideraciones extraídas del análisis del mismo.
- Vista: en él se englobarán aquellas clases correspondientes a la interacción física entre el usuario y el sistema, teniendo en cuenta la posibilidad de desarrollar interfaces gráficas de usuario.
- Controlador: las clases que se enmarquen dentro de este módulo estarán asociadas con la idea de "bound-dispatcher" con el que se relaciona el Controlador, encargado de gestionar simultáneamente la comunicación entre el modelo de negocio (módulo Modelo) y el usuario (módulo Vista). Su misión consiste en dirigir el control interno de la aplicación, gestionar los mensajes recibidos desde los componentes del módulo Vista, verificarlos, adaptarlos al modelo conceptual y aplicar las reglas del Modelo (modelo de negocio) definidas en cada caso.

En el presente proyecto, se decide además que sea la capa del controlador la encargada de la lectura y escritura de los patrones en los ficheros OWL, para aislar de esta funcionalidad a la capa del modelo, sin querer dotar a la vista de otra funcionalidad distinta a la de la interacción física entre el usuario y el sistema.

Finalmente, se exponen y razonan las ventajas y los inconvenientes de la arquitectura empleada MVC:

Ventajas:

- Resulta un modelo sencillo de entender, ya que la separación del diagrama global de clases en módulos favorece la comprensión del sistema.

- Minimiza las dependencias entre módulos o partes del sistema, y además fomenta la reutilización, posibilitando por ejemplo que el mismo modelo de negocio pueda emplearse en otros sistemas que cuenten con otros tipos de interacción "usuario-sistema" (cambiar interfaz sin tocar el resto).

Inconvenientes:

- No es una arquitectura que pueda considerarse "pura" ya que las dependencias relacionan varias capas, pudiéndose acceder desde el módulo Vista al módulo Modelo, como así ha ocurrido en el presente proyecto. La ilustración 5 muestra un dibujo aclaratorio de esta arquitectura:

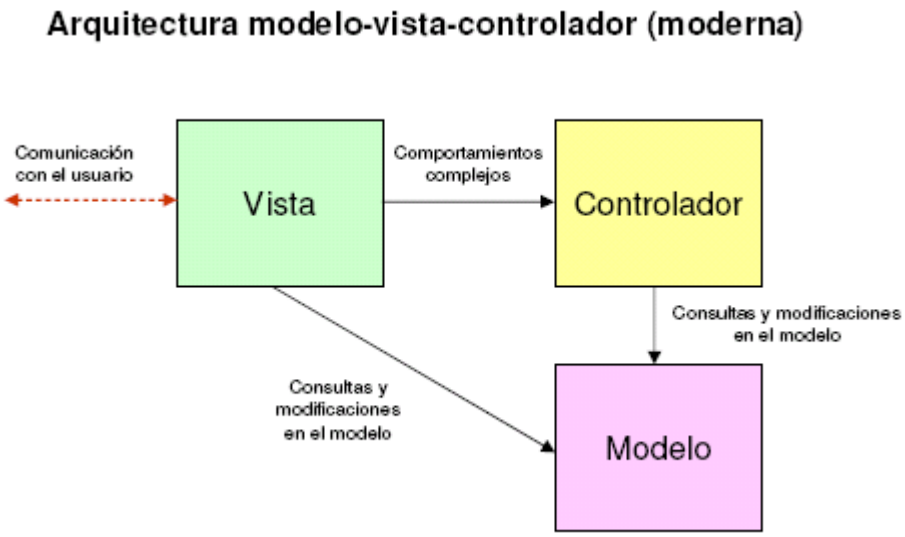


Ilustración 5: Arquitectura MVC

RECURSOS

El siguiente paso consistía en definir la forma de almacenar los patrones usados por la aplicación. Por un lado, había que definir una estructura para su almacenamiento persistente en OWL, y por el otro, también era necesario definir una estructura de clases Java, para almacenar dichos patrones durante la ejecución de la aplicación.

En los siguientes apartados se exponen ambos diagramas.

Diagrama de clases de patrones para gestión por aplicación

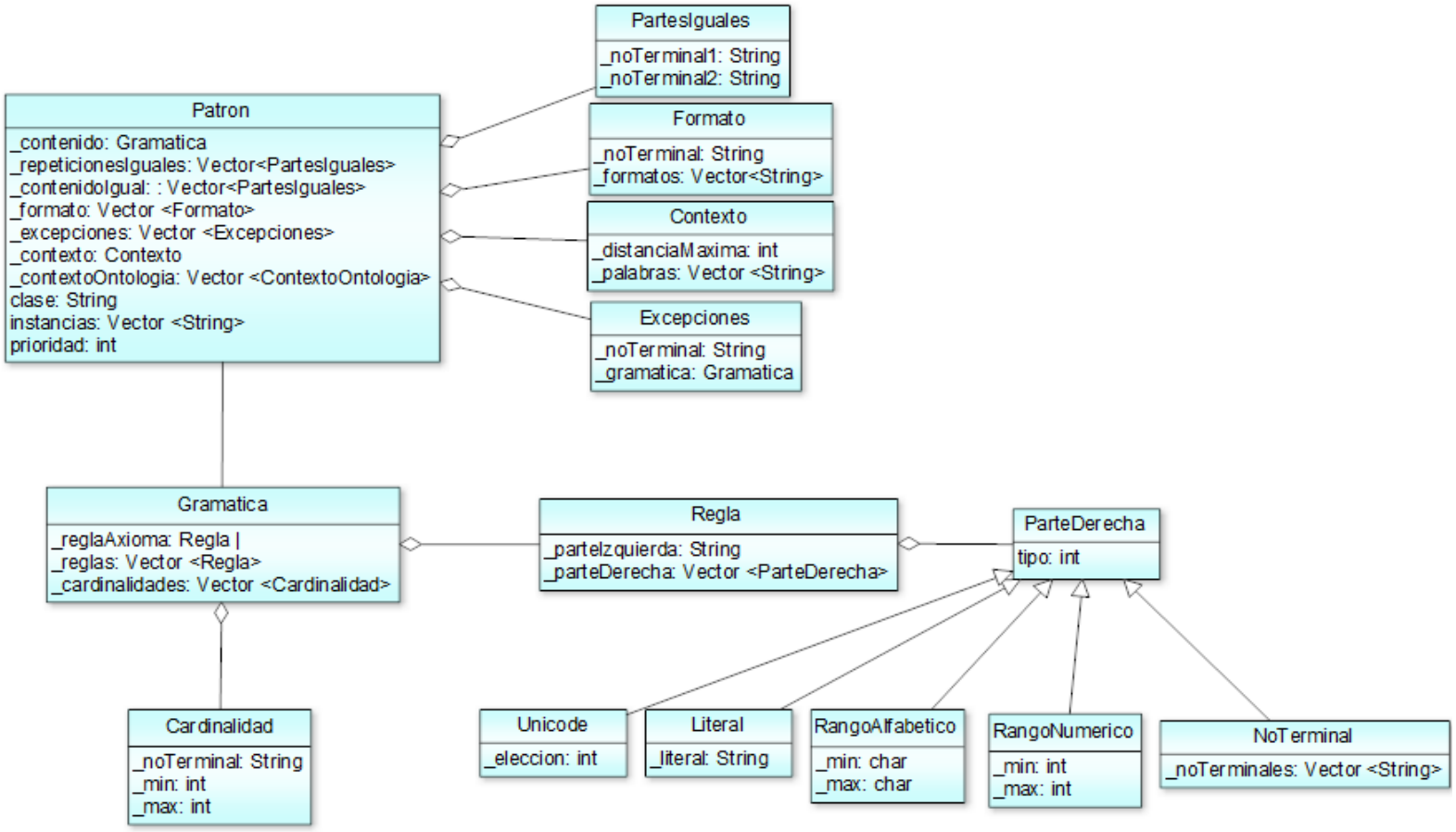


Ilustración 6: Diagrama de clases

En el diagrama de la ilustración 6, se puede comprobar cómo refleja con exactitud la definición de patrón especificada en los requisitos. Esto es, un patrón se compone de una gramática que define la estructura de la entidad, sumada a una serie de condicionantes, creados para la adición de restricciones de formato o contexto, o características especiales como la repetición de la cardinalidad o el contenido de dos no terminales de la gramática asociada, lo que conllevará a excluir del lenguaje palabras que sí eran reconocidas. Estos condicionantes se almacenan dentro de la clase Patron en vectores de una serie de clases creadas para almacenar la información necesaria de la restricción. Estas clases son las siguientes:

- **PartesIguales**: su cometido es el de almacenar los dos símbolos de la gramática que han de tener el mismo contenido, o la misma cardinalidad. Nótese cómo se ha reutilizado la misma clase para albergar dos tipos distintos de condicionantes, dada la similitud entre ellos.
- **Formato**: almacena para cada símbolo de la gramática todos los formatos definidos para dicho no terminal, en caso de haberse definido alguno.
- **Contexto**: almacena todas las palabras que han de estar presentes en el contexto de la entidad, y las distancia máxima que puede haber entre ellas.
- **Excepciones**: asocia a un símbolo del patrón una gramática, que desechará la entidad en caso de que el contenido asociado al símbolo tratado satisfaga la gramática almacenada. Se puede apreciar que las excepciones introducidas trabajan con gramáticas y no con patrones, con el objetivo de simplificar su definición. El caso más corriente de excepción sería el de la exclusión de una palabra. Para su definición, habría que asignar al axioma de la gramática definidora del patrón, una excepción consistente en una gramática muy simple (Axioma → "palabra a excluir del patrón que estamos definiendo")

Por otro lado, la gramática no es más que una serie de reglas, sumadas a la definición de una serie de cardinalidades para cada símbolo de ésta. Dentro de la clase, existirán los métodos necesarios para la verificación de que se está trabajando con una gramática bien formada.

Por último, una regla de la gramática relaciona un símbolo no terminal de la gramática (parte izquierda) con un conjunto de opciones (parte derecha). Cada una de estas opciones estarían separadas por el operador "OR". Cada opción puede ser de la siguiente forma:

- Literal: el contenido asociado al símbolo ha de ser una secuencia de caracteres invariante.
- RangoAlfabetico: el contenido es un carácter, dentro de unos límites (rango máximo y rango mínimo).
- RangoNumerico: el contenido es un número de una cifra, dentro de unos límites(rango máximo y rango mínimo).
- Unicode: el contenido es un carácter, de un determinado rango Unicode preestablecido:
    - Carácter
    - Letra
    - Puntuación de apertura
    - Puntuación de cierre
    - Otra puntuación
    - Separador
    - Otro símbolo
- NoTerminal: para satisfacer la regla, se ha de satisfacer cada una de las reglas asociadas a cada símbolo no terminal aquí reseñado, y en el orden establecido.

Un patrón también tiene definidos la clase a la que pertenece y las entidades que desde su creación ha capturado, y el usuario ha decidido utilizar. Por último, un patrón posee una determinada prioridad, que es la que condiciona si una entidad es capturada por el patrón o no, en caso de que dos patrones lanzados en la misma búsqueda capturen la misma entidad.

Diagrama clases patrones para almacenamiento OWL

La ilustración 7 presenta el diagrama OWL elaborado para el almacenamiento de los patrones en OWL:

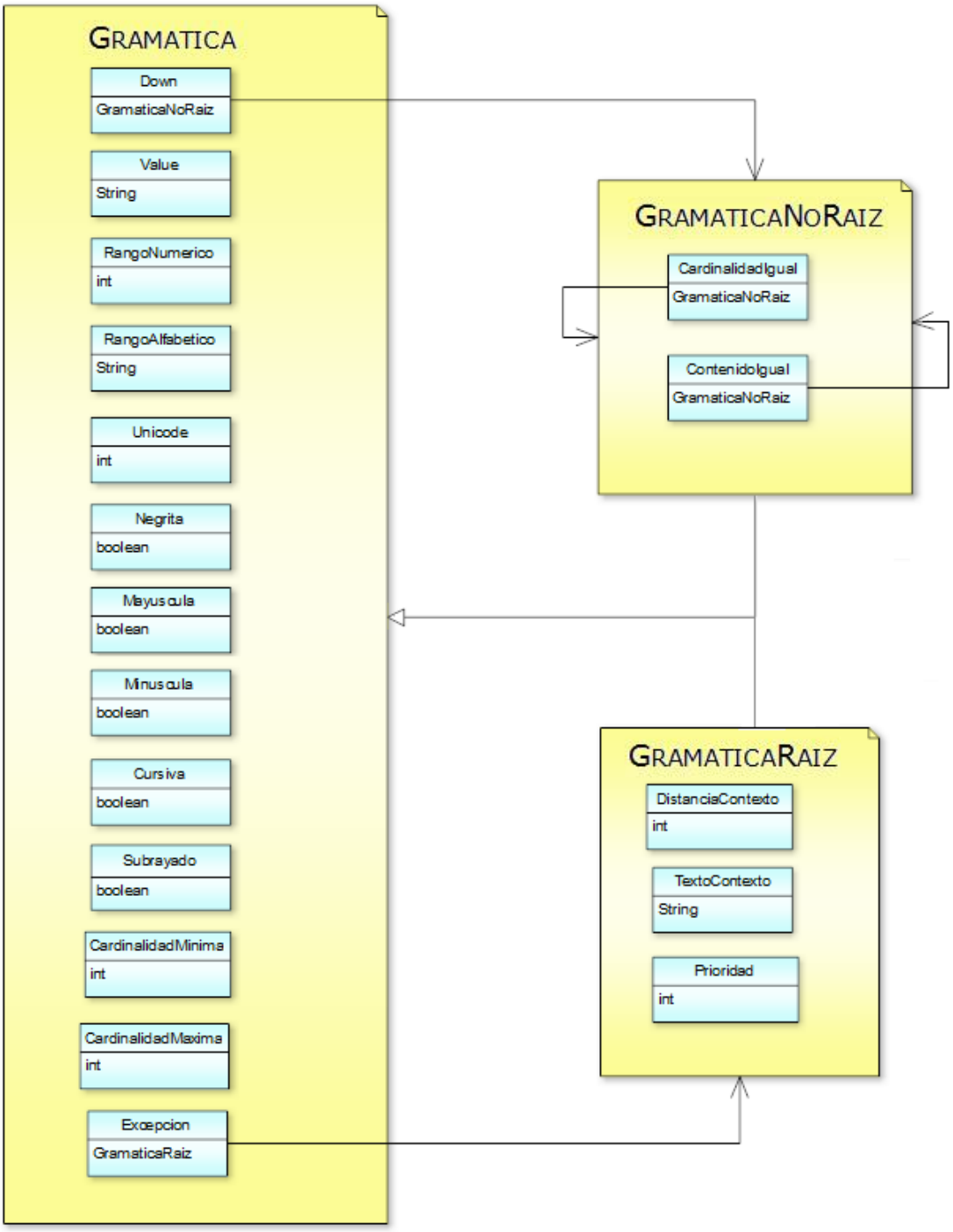


Ilustración 7: Diagrama OWL

Los recuadros amarillos representan las clases, mientras que los azules representan las propiedades presentes en cada una de las clases, apareciendo el rango de cada propiedad en la parte inferior del mismo.

Como podemos apreciar en el diagrama, para el almacenamiento de los patrones en OWL se han diseñado tres clases:

- Gramatica (abstracta)
- GramaticaRaiz
- GramaticaNoRaiz

La clase abstracta Gramatica se ha diseñado para contener las propiedades comunes a las clases GramaticaRaiz y GramaticaNoRaiz, pues ambas son subclases de la primera.

La clase GramaticaRaiz se concibe para representar el axioma o término raíz de la gramática, junto con todas sus propiedades exclusivas a esta clase. Propiedades solo aplicables al axioma de la gramática son el *contexto* y la *prioridad*. Realmente, ambas prioridades lo son del patrón en sí, y por definición de la gramática, pero se ha decidido incluirlas en la clase que representa el axioma de la misma por simplicidad. Esto es posible debido a que una gramática tiene uno y sólo un axioma.

La clase GramaticaNoRaiz se crea para representar aquellos símbolos no terminales de la gramática que no se corresponden con el axioma. Por tanto, dicha clase albergará las propiedades de la gramática no aplicables al axioma. Estas se corresponden con la restricción presente en el patrón referente a que dos símbolos no terminales de la gramática presenten la misma cardinalidad o el mismo contenido.

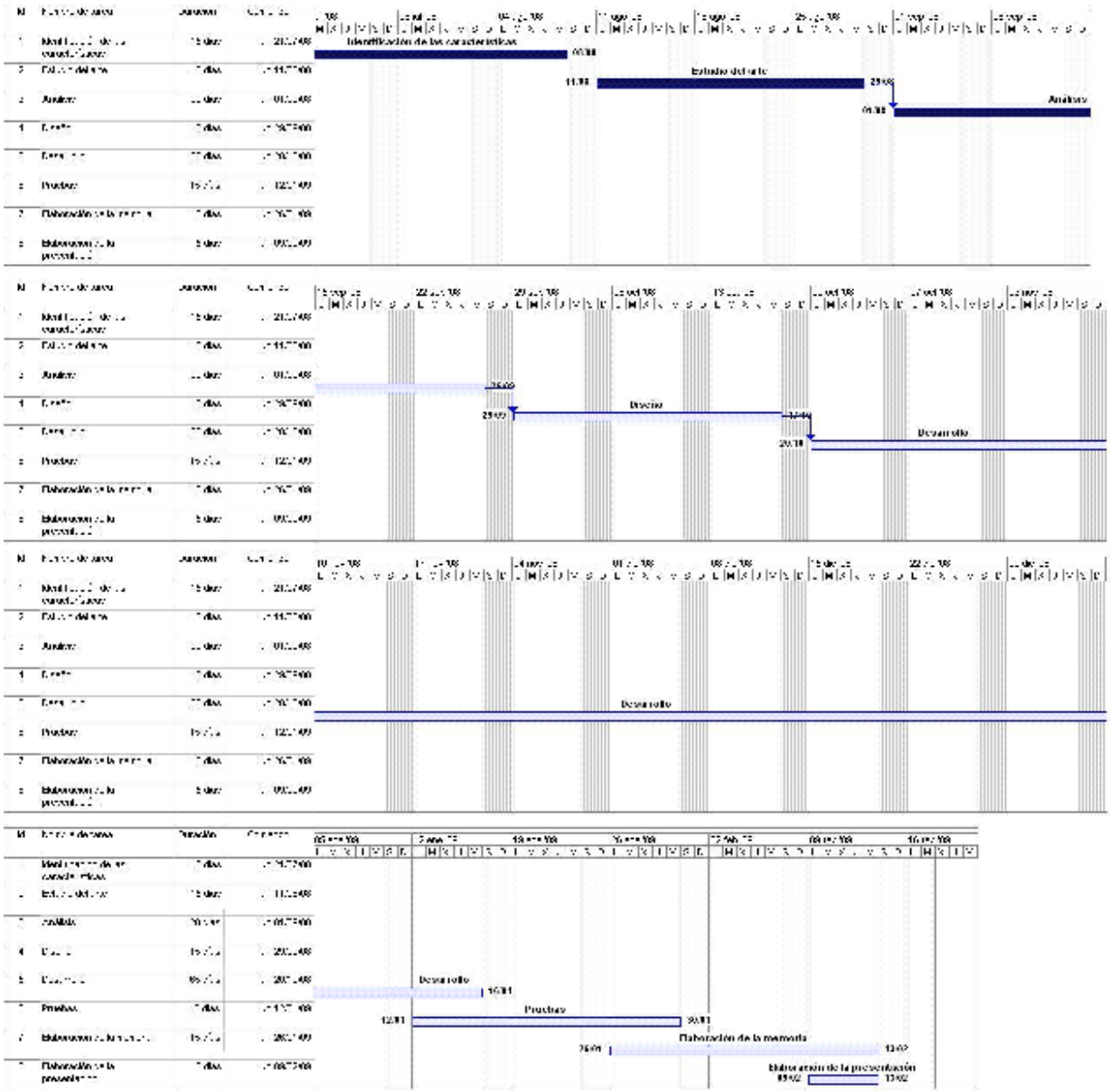
Podemos concluir que la clase Gramática alberga las propiedades que pueden presentar un símbolo no terminal de la gramática, sin diferenciaciones entre el axioma y el resto. Por ejemplo, la propiedad *negrita* es aplicable a todos los no terminales de la gramática, puesto que podemos querer forzar que una parte de la entidad esté presente en negrita, o por el contrario toda la entidad lo esté, caso en el cual, el axioma presentaría la propiedad *negrita.*

Nótese cómo, al contrario que en el de clases, esta estructura no diferencia entre la gramática y los condicionantes. Para su almacenamiento en OWL, por simplicidad, se ha decidido albergar cada condicionante dentro del símbolo de la

gramática con el que se corresponde, aprovechando de una manera más eficiente la potencia que proporciona las tripletas RDF en OWL.

PLANIFICACIÓN

A continuación se expone el Diagrama de Gantt relativo a la planificación del proyecto. Se ha de tener en cuenta que el número de horas dedicadas al presente proyecto oscila en torno a las 6 horas diarias.



IMPLEMENTACIÓN

A continuación, se exponen una serie de datos relativos al código elaborado, para hacer una ligera idea al lector de sus propiedades, sin querer entrar en los detalles de éste.

Vista	
Número de clases	26 clases
Líneas de código	5.668 líneas
Kb de código	315 Kb

Tabla 1: Detalles código capa Vista

Controlador	
Número de clases	3 clases
Líneas de código	3.351 líneas
Kb de código	162 Kb

Tabla 2: Detalles código capa Controlador

Modelo	
Número de clases	22 clases
Líneas de código	2.626 líneas
Kb de código	78,4 Kb

Tabla 3: Detalles código capa Modelo

Sistema completo	
Número de clases	51 clases
Líneas de código	11.645 líneas
Kb de código	555,4 Kb

Tabla 4: Detalles código totales

Como se puede apreciar en las tablas anteriores, el módulo de la herramienta con mayor carga de trabajo es la capa de la vista, puesto que la aplicación se compone de varias interfaces que tienen como objetivo la interacción con el usuario para el manejo de ontologías y patrones, además de mostrar los resultados de una manera visible y ordenada.

Nótese como la capa de Controlador, a pesar de no disponer de un gran número de clases, si que es extensa en cuanto a número de líneas de código se refiere, puesto que se encarga de la comunicación entre las capas de Vista y Modelo y además es la encargada de la lectura y escritura de las ontologías, debido a que se trató de independizar al resto de capas de esta funcionalidad.

5 FUNCIONAMIENTO Y RESULTADOS

En este apartado se pretende, por un lado, mostrar al lector cómo funciona la herramienta, a la vez que se presentan los resultados obtenidos en la captura de entidades, a partir de los patrones generados mediante un ejemplo de uso.

SELECCIÓN DEL CORPUS

La ilustración 8 muestra la pantalla que aparece al inicializar la herramienta.

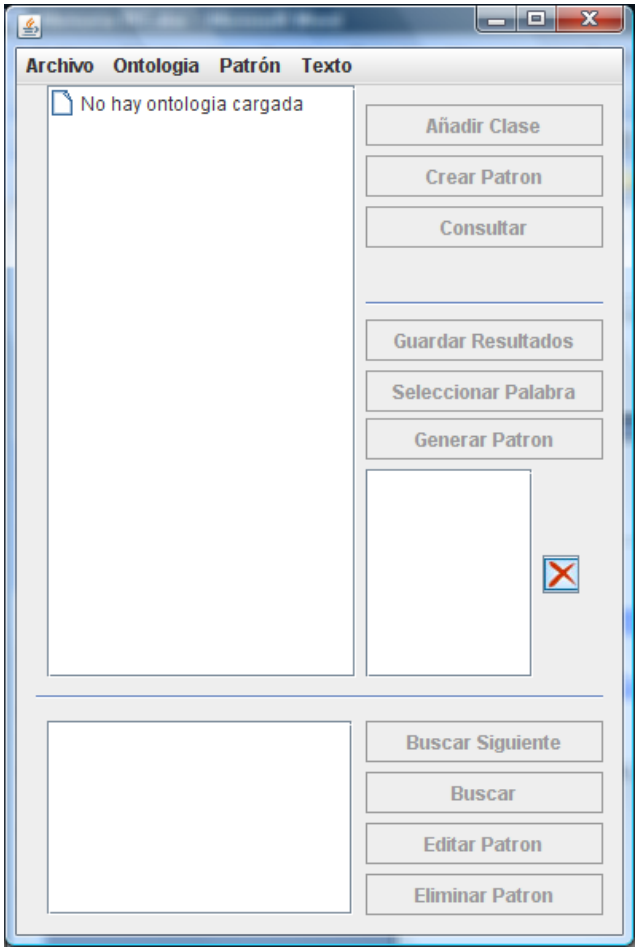


Ilustración 8: Interfaz principal

Si se pincha en "*Archivo*", aparecerá la opción de "*Cargar Texto*":

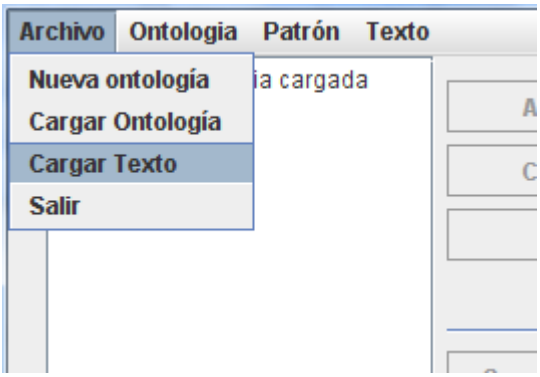


Ilustración 9: Menú cargar texto

Si se pincha sobre ella, nos aparecerá un diálogo para seleccionar un archivo de texto almacenado en el sistema, tal y como muestra la ilustración 10.

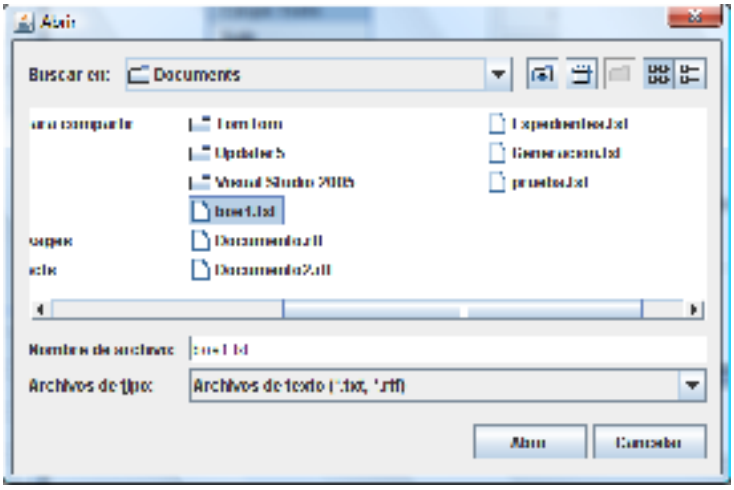


Ilustración 10: Selección del corpus

Seleccionamos el archivo deseado, y al pulsar en "Abrir", se cargará en la herramienta. Este archivo se podrá utilizar a partir de este momento, para capturar palabras para la generación automática de patrones, o para capturar entidades de patrones cargados en la herramienta.

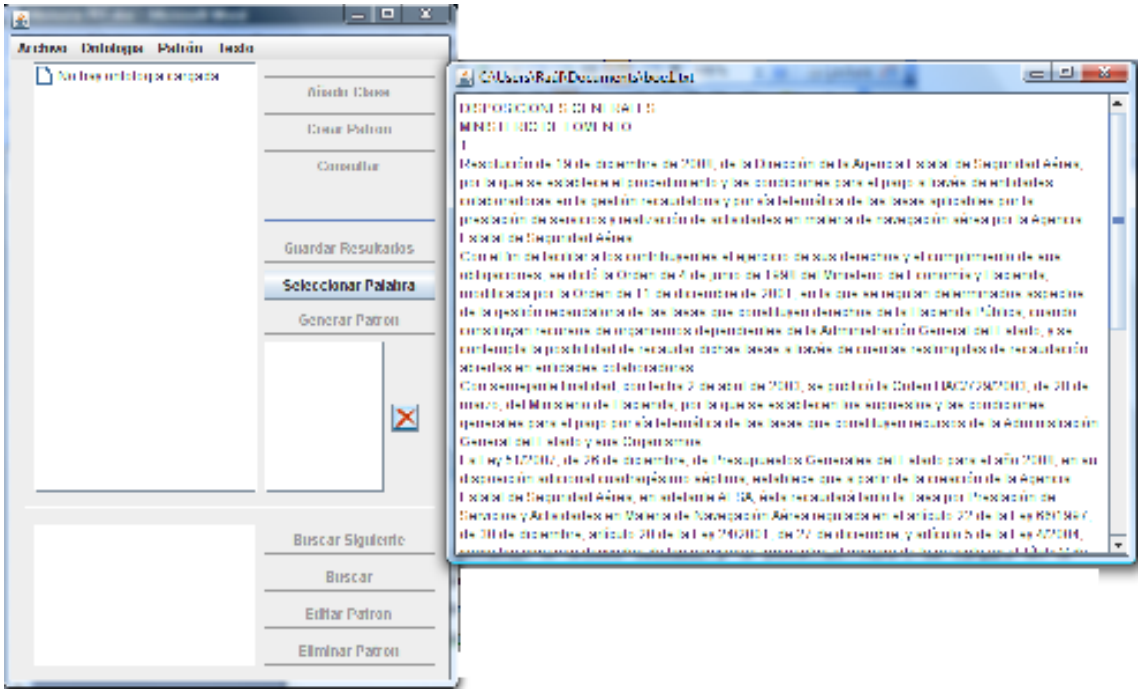


Ilustración 11: Interfaz aplicación

Para la obtención de resultados, se cargarán en la herramienta, como ejemplo, distintos documentos procedentes del Boletín Oficial del Estado (BOE), pues son documentos ricos en entidades reconocidas por patrones, como pueden ser los identificadores de las Leyes Orgánicas. Para aumentar esta riqueza, los documentos elegidos serán de distintas áreas.

CREACIÓN DE UNA ONTOLOGÍA NUEVA

La herramienta da la posibilidad de trabajar con ontologías existentes, pero también da la posibilidad de crear una nueva ontología vacía, de manera que el usuario pueda rellenarla con las clases e instancias que desee. Para ello, se deberá pinchar en "Archivo" y posteriormente en "*Nueva Ontología*":

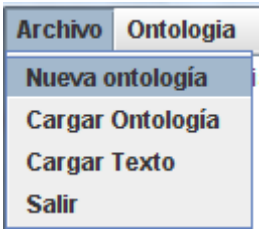


Ilustración 12: Menú nueva ontología

La ilustración 13 muestra el diálogo, en el que, tras seleccionar la carpeta en la que se almacenará la ontología, el usuario deberá introducir el nombre de la ontología:

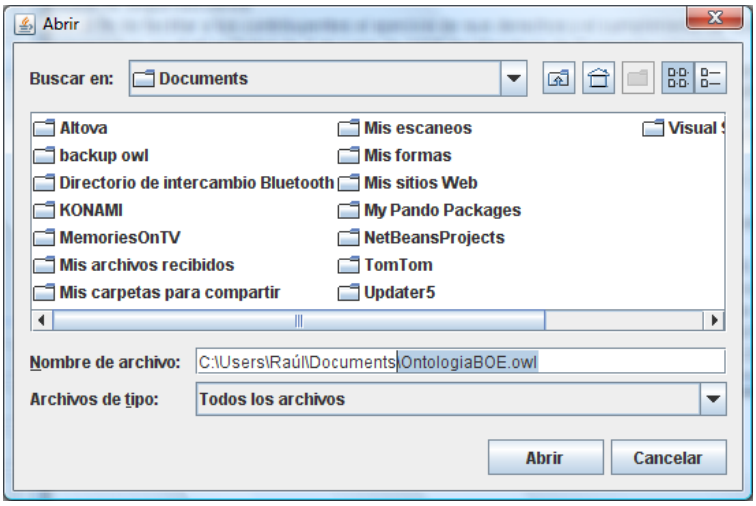


Ilustración 13: Selección ubicación nueva ontología

Ya tenemos la ontología cargada en la aplicación. Ahora sobre ella se pueden crear clases (y subclases). Para ello, se deberá pinchar en "*Añadir Clase*" (es necesario tener seleccionado la clase padre):

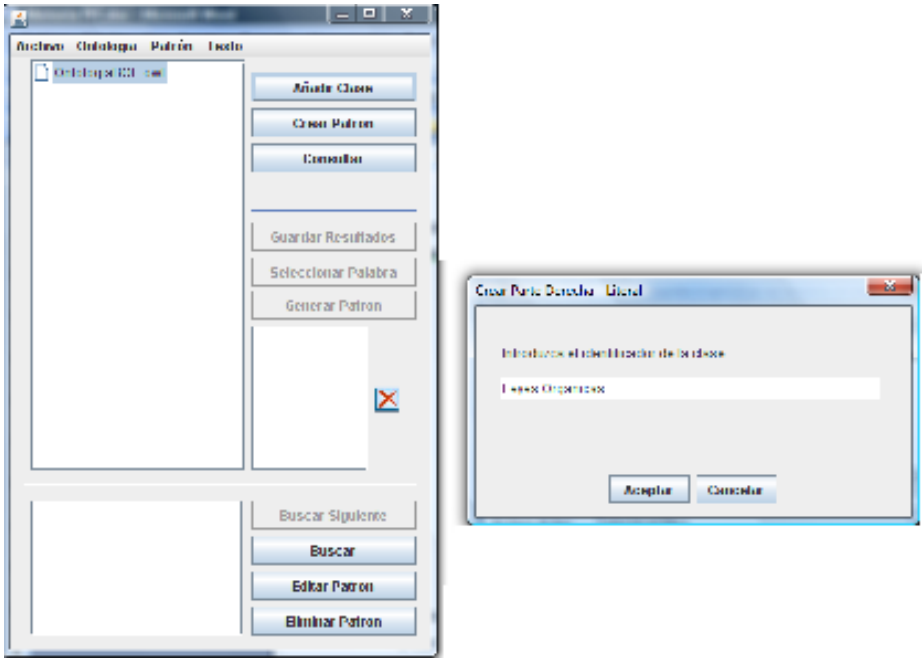


Ilustración 14: Interfaz crear nueva clase

Ya tenemos, pues la clase creada, para definir patrones sobre ella.

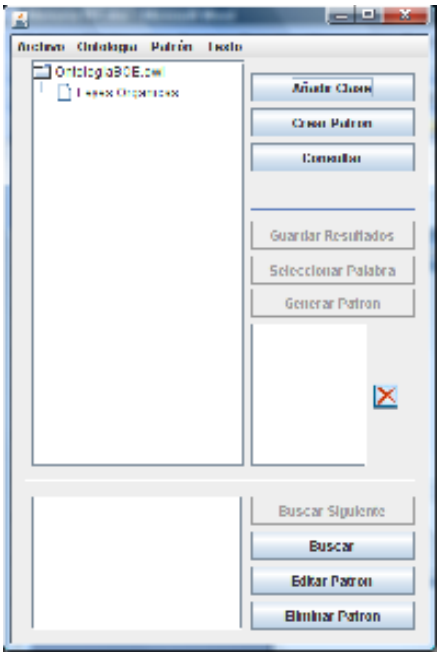


Ilustración 15: Interfaz con clase creada

Para la obtención de resultados, se creará una nueva ontología (ontología de patrones), debido a que los patrones a utilizar no existen y han de ser creados. Se crearán las clases correspondientes dentro de la ontología para albergar estos patrones creados.

CARGAR UN ONTOLOGÍA EXISTENTE

Puede ocurrir que el usuario quiera trabajar sobre una ontología almacenada en su sistema de ficheros, bien porque dispone de ella (procedente de la Web), bien porque es una ontología que había creado en otro momento y quiere recuperarla de nuevo, para seguir trabajando sobre ella. Para ello, deberá pinchar en "*Archivo*", y posteriormente en "*Cargar Ontología*".

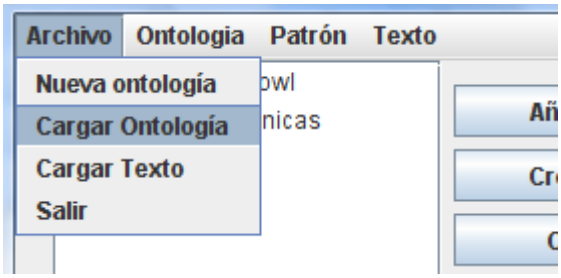


Ilustración 16: Menú cargar ontología

Aparecerá un diálogo, en el que escogeremos la ontología que deseamos cargar en la herramienta:

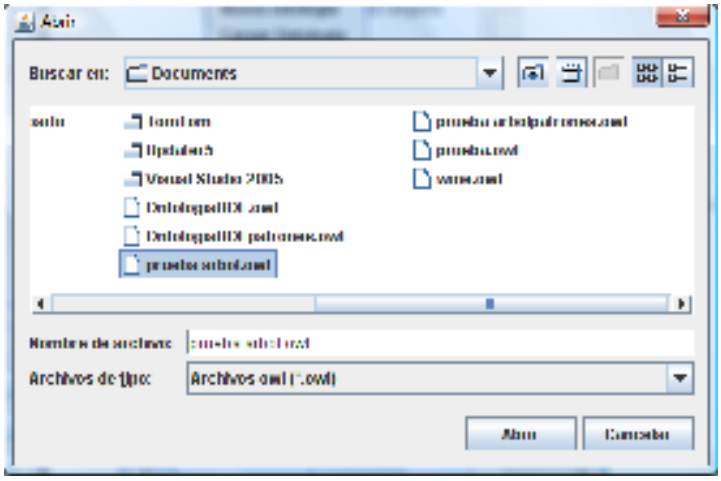


Ilustración 17: Selección nueva ontología

La ontología cargada se mostrará en la aplicación, con todas sus clases e instancias almacenadas en ella:

Ilustración 18: Visor de ontologías

GENERACIÓN AUTOMÁTICA DE PATRONES

Una vez cargados en la aplicación tanto el corpus de entrenamiento como la ontología de usuario, la aplicación da la posibilidad de generar automáticamente patrones a partir de una serie de ejemplos, escogidos del corpus. Lo primero que habría que hacer seria escoger una serie de ejemplos de entidades del patrón que queremos generar, pulsando en el botón "*Seleccionar Palabra*", tras seleccionar dentro del corpus la palabra deseada.

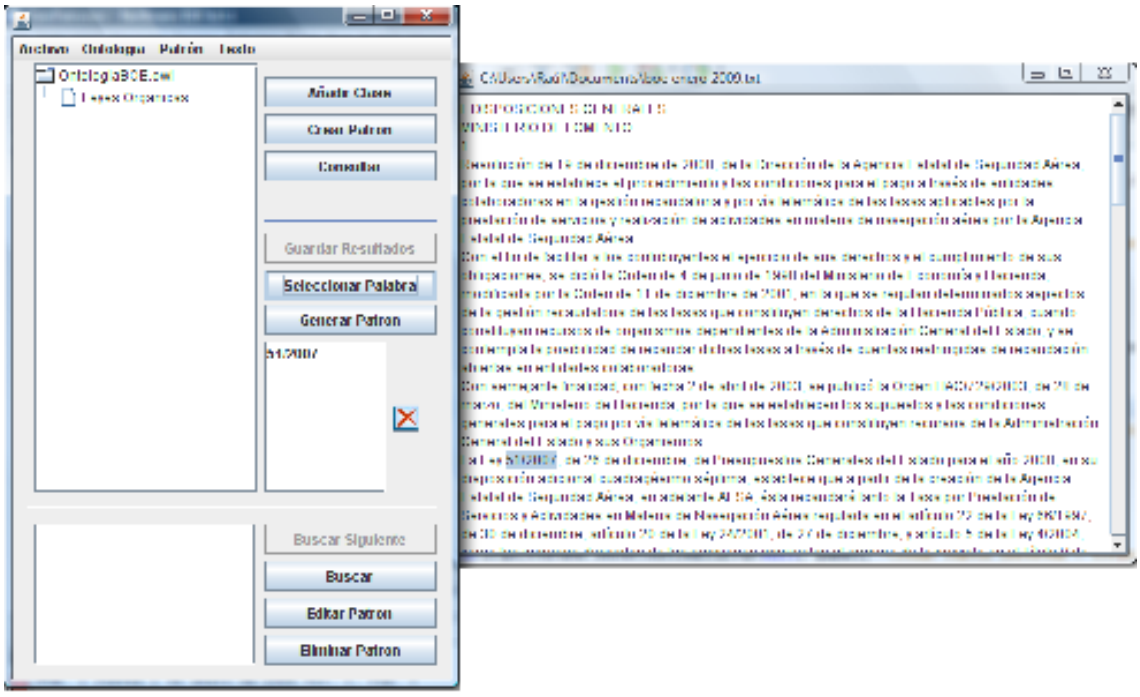


Ilustración 19: Selección de palabras

Las palabras seleccionadas aparecerán en el recuadro situado en la parte derecha de la interfaz principal:

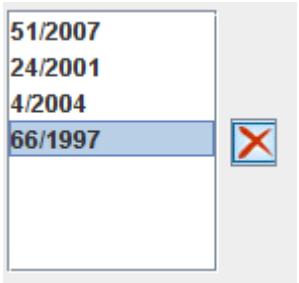


Ilustración 20: Visor de palabras seleccionadas

Cuando se hayan escogido todas las entidades a partir de las cuales se desea generar un nuevo patrón, se deberá hacer clic en "*Generar Patron*". Se generará un

patrón reconocedor de las entidades seleccionadas, que se almacenará en la ontología en la clase seleccionada. En este ejemplo se ha seleccionado que se almacene en la clase "Leyes Orgánicas".

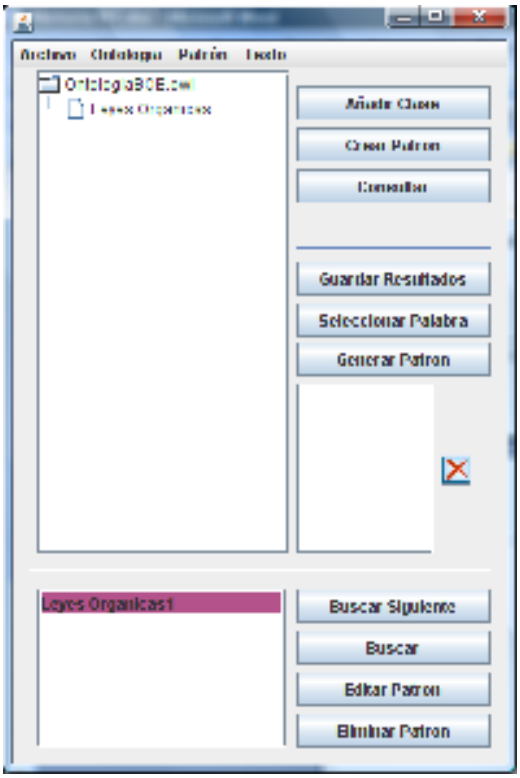


Ilustración 21: Nuevo patrón generado

Este patrón quedará almacenado en la ontología, y podrá ser usado para lanzar búsquedas. Asimismo, podrá ser editado posteriormente por el usuario a fin de pulir detalles, para hacer más eficiente la captura de entidades.

CREACIÓN Y EDICIÓN DE PATRONES

Puede ser que el usuario desee crear un nuevo patrón partiendo desde cero, pues tiene bien claras sus especificaciones, para el reconocimiento de entidades. Por otro lado, un patrón, sea creado de manera automática, o de manera manual, puede ser posteriormente editado. Tanto la edición de patrones, como la creación manual de éstos, comparten la misma interfaz. Para la creación de patrones de manera manual deberemos seleccionar la clase dentro de la ontología en la que se desee almacenar el nuevo patrón y hacer clic en "*Crear Patron*". La ilustración 22 muestra la interfaz que aparece:

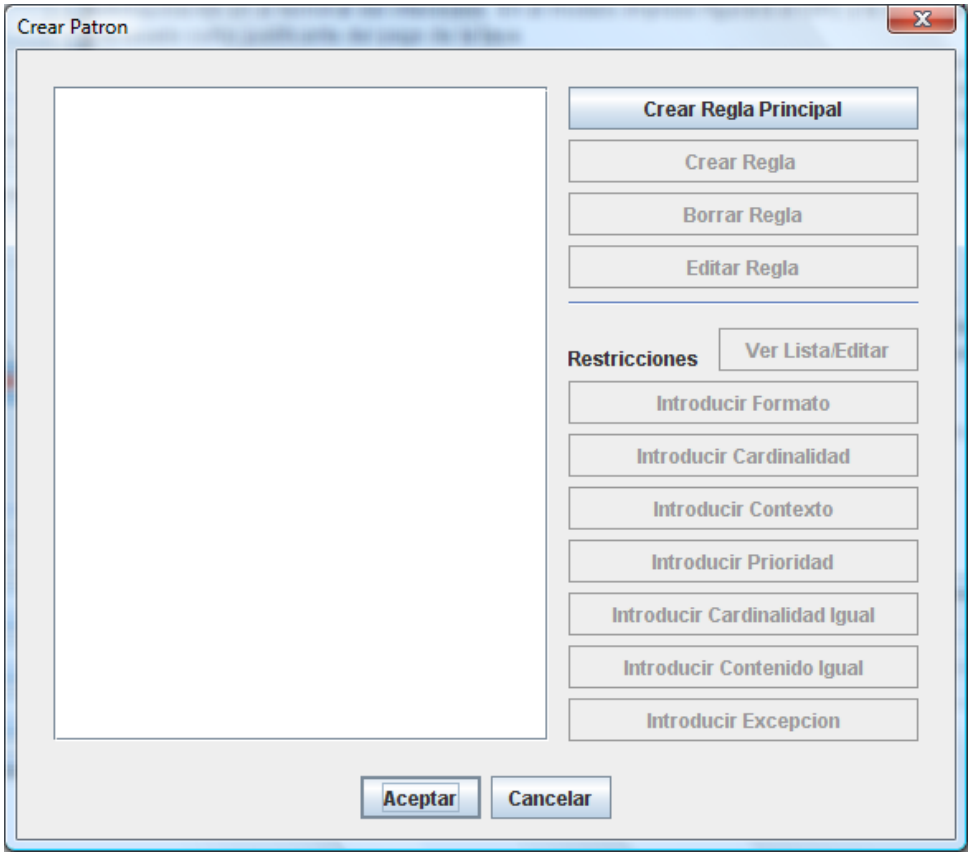


Ilustración 22: Interfaz creación/edición de patrones

Una vez se vaya creando la gramática relativa al patrón, ésta se irá mostrando en la parte izquierda de la interfaz. Lo primero que hay que realizar para crear un nuevo patrón es introducir la regla relativa al axioma, o regla principal (haciendo clic en "*Crear Regla Principal*"):

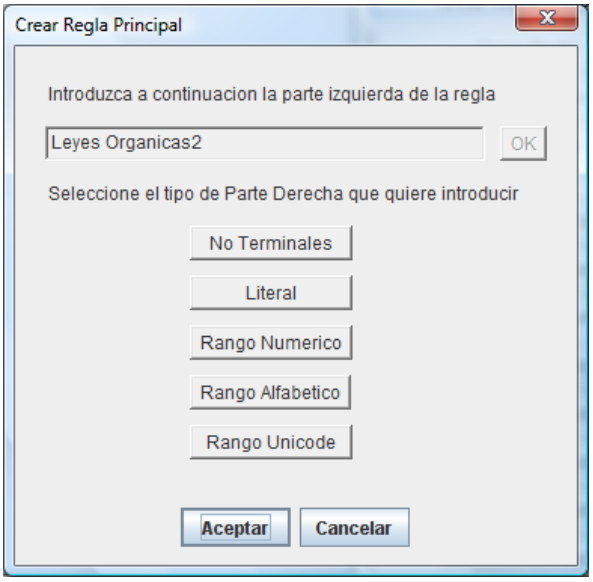


Ilustración 23: Interfaz creación de reglas

La ilustración 23 muestra la interfaz con la que se crean nuevas reglas en la gramática asociada al patrón. En este caso, al haber seleccionado "*Crear Regla Principal*", nos aparece directamente (e invariante) el no símbolo no terminal al que está asociado la regla (el axioma en este caso). Si se hubiera hecho clic en "*Crear Regla*", el usuario hubiera tenido que introducir en el cuadro de texto situado en la parte superior de la ventana el identificador del símbolo no terminal asociado a la regla y hacer clic posteriormente en "*OK*". Como se puede apreciar en la anterior imagen, existen cinco posibles modos de introducir la parte derecha de la regla (cada una con su botón correspondiente)

- No Terminales: si se desea que el símbolo de la parte izquierda de la regla derive en una serie de símbolos, también no terminales. La ilustración 24 muestra la interfaz.

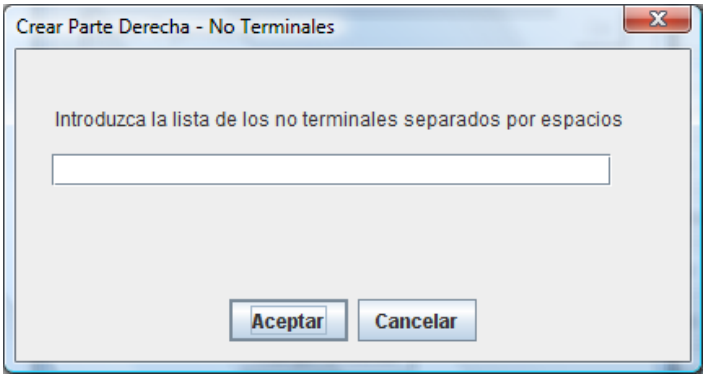


Ilustración 24: Interfaz creación parte derecha (No Terminal)

En la que el usuario debe introducir la serie de símbolos no terminales en las que desee que derive la regla, separados por espacios.

- Literal: si se desea que se derive en una serie de caracteres invariantes. La ilustración 25 muestra la interfaz.

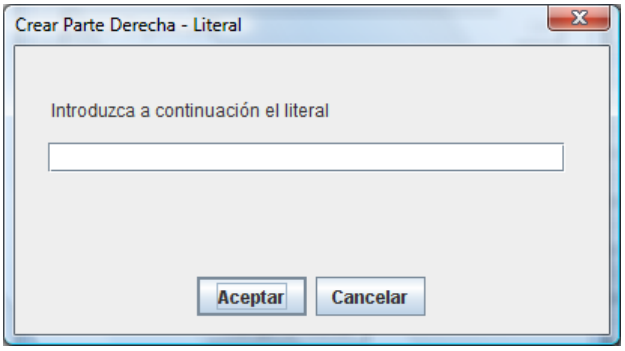


Ilustración 25: Interfaz creación parte derecha (Literal)

El usuario tendría que introducir el literal en al cuadro de texto.

- Rango Numérico: si se desea que el no terminal derive en un dígito, comprendido entre un dígito mínimo y uno máximo. La ilustración 26 muestra la interfaz.

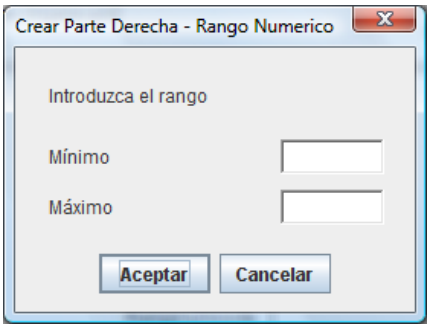


Ilustración 26: Interfaz creación parte derecha (Rango Numérico)

- Rango Alfabético: si se desea que el no terminal derive en un carácter, comprendido entre un carácter mínimo y uno máximo. La ilustración 27 muestra la interfaz.

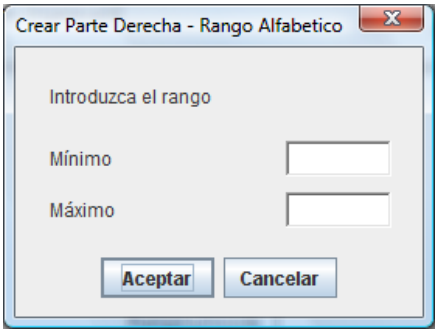


Ilustración 27: Interfaz creación parte derecha (Rango Alfabético)

- Unicode : si se desea que el no terminal derive en un carácter de un rango Unicode determinado. La interfaz se muestra en la ilustración 28:

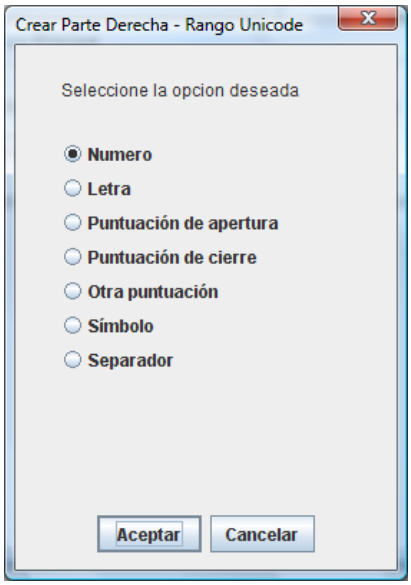


Ilustración 28: Interfaz creación parte derecha (Rango Unicode)

El usuario seleccionaría el rango Unicode asociado a la regla.

Una regla puede tener tantas cláusulas en la parte derecha como el usuario desee. Este conjunto de cláusulas se introducen separadas por el operador "OR". Para ello, iría introduciéndolas en la interfaz de la Ilustración 20. Por ejemplo, si el usuario deseara crear una regla con una cláusula del tipo "Literal" y otra del tipo "Rango Unicode", tendría que hacer clic en la citada interfaz en "*Literal*", introducir el literal deseado y posteriormente hacer clic en "*Rango Unicode*" para seleccionar el rango deseado. Una vez finalizado, en la citada interfaz habría que hacer clic en "*Aceptar*". La regla recién creada aparecerá en la parte izquierda de la ventana, como se pude apreciar en la ilustración 29.

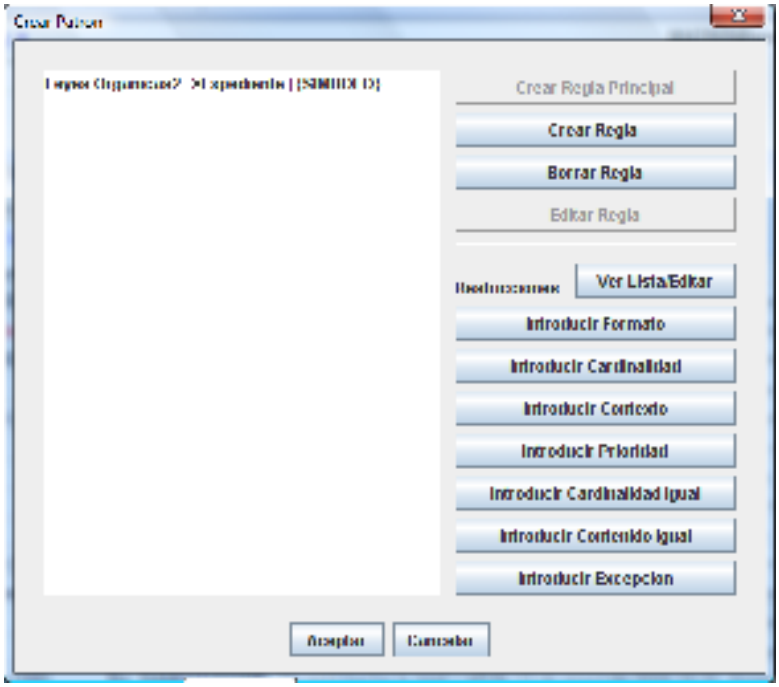


Ilustración 29: Regla creada correctamente

Asimismo, se puede comprobar como otros botones de la interfaz se han habilitados al haber creado la regla relativa al axioma de la gramática.

CAPTURA DE ENTIDADES

Este apartado tiene el objetivo de comprobar los resultados que obtiene la herramienta al lanzar una búsqueda sobre el texto cargado de una serie de patrones. Para ello, se seguirán los siguientes pasos:

- Se cargará el texto relativo al BOE del Ministerio de Industria Turismo y Comercio de Diciembre de 2008[BOE1].
- Se creará una ontología nueva, a la que llamaremos OntologiaBOE.OWL y crearemos sobre ella las siguientes clases en la raíz:
  - Identificadores Legales: en ella crearemos las siguientes subclases:
    - Leyes orgánicas
    - Decretos
    - Órdenes
  - Fechas
  - Prueba

Seleccionamos sobre el texto cargado las siguientes entidades y generamos automáticamente un patrón, llamado Ordenes1, destinado a capturar las órdenes ministeriales presentes en el texto:

- ITC/3802/2008
- ITC/3863/2007

Los resultados obtenidos se muestran en la tabla 5:

Ordenes1	
ITC/3802/2008	ITC/3862/2007
ITC/3863/2007	ITC/3863/2007
Precisión	4/4=100%
Recall	4/22=18,1%

Tabla 5: Resultados obtenidos por el patrón Ordenes1

Podemos comprobar cómo el patrón es muy restrictivo, pues sólo ha capturado las entidades introducidas como ejemplos, junto con otra muy similar. Esto ha sucedido porque las entidades escogidas como ejemplos no han sido significativas, y el patrón ha entendió que todas las entidades comenzarían por "ITC/38". Para corregir este aspecto, creamos el patrón Ordenes 2 añadiendo como ejemplo la entidad "ITC/2857/2008", obteniéndose los resultados mostrados en la tabla 6.

Ordenes2	
ITC/3802/2008	ITC/3993/2006
ITC/3863/2007	ITC/3993/2006
ITC/2857/2008	ITC/4099/2005
ITC/3126/2005	ITC/3995/2006
ITC/2857/2008	ITC/3993/2006
ITC/3995/2006	ITC/2308/2007
ITC/3995/2006	ITC/2308/2007
ITC/3995/2006	ITC/3126/2005
ITC/3862/2007	ITC/3126/2005
ITC/3863/2007	
Precisión	19/19=100%
Recall	19/22=86,3%

Tabla 6: Resultados obtenidos por el patrón Ordenes2

De las 22 entidades presentes en el texto, el patrón solo ha dejado sin capturar las siguientes:

- ECO/2692/2002
- ECO/302/2002
- ECO/2692/2002

Obviamente esto sucede porque con los ejemplos introducidos para la generación del patrón, es imposible que éste conociera que existen órdenes que comienzan por el identificador "ECO", pues todos los ejemplos comenzaban con el identificador ITC. Si se añade como ejemplo la entidad ECO/302/2002, los resultados obtenidos por el patrón generado (Ordenes3) se muestran en la tabla 7.

Ordenes3	
ITC/3802/2008	ITC/3862/2007
ITC/3863/2007	ITC/3863/2007
ITC/2857/2008	ITC/3993/2006
ECO/2692/2002	ITC/3993/2006
ECO/302/2002	ITC/4099/2005
ITC/3126/2005	ITC/3995/2006
ITC/2857/2008	ITC/3993/2006

ECO/2692/2002	ITC/2308/2007
ITC/3995/2006	ITC/2308/2007
ITC/3995/2006	ITC/3126/2005
ITC/3995/2006	ITC/3126/2005
Precisión	22/22=100%
Recall	22/22=100%

Tabla 7: Resultados obtenidos por el patrón Ordenes3

Ha quedado demostrado cómo el patrón generado mejora los resultados cuanto más representativos son los ejemplos introducidos.

A continuación se pasa a generar manualmente el patrón. La generación manual tiene la ventaja de partir de la sabiduría del usuario, el cual conoce todas las posibilidades de entidades que existes. Por ejemplo, en el caso que nos ocupa, conoce que existen dos tipos de órdenes, las que comienzan por "*ITC*", y las que comienzan por "*ECO*". Se generó el siguiente patrón:

S → A B C D

A → "ITC/" | "ECO/"

B → {dígito}

C → "/200"

D → {digito}

Además se introdujeron las siguientes condiciones:

- B → Cardinalidad mínima = 2 y Cardinalidad mínima = 4
- Contexto → Debe aparecer la palabra "Orden"

El patrón generado se denominó Ordenes4, y por el convenio de nombrado adoptado, los símbolos no terminales introducidos son llamados como el patrón, seguido de un dígito secuencial (Ordenes41, Ordenes42, etc.). En la memoria se expondrán identificadores compuestos de una sola letra, con el objetivo de simplificar la comprensión al lector.

Los resultados obtenidos se exponen en la tabla 8:

Ordenes4	
ITC/3802/2008	ITC/3862/2007
ITC/3863/2007	ITC/3863/2007
ITC/2857/2008	ITC/3993/2006
ECO/2692/2002	ITC/3993/2006
ECO/302/2002	ITC/4099/2005
ITC/3126/2005	ITC/3995/2006
ITC/2857/2008	ITC/3993/2006
ECO/2692/2002	ITC/2308/2007
ITC/3995/2006	ITC/2308/2007
ITC/3995/2006	ITC/3126/2005
ITC/3995/2006	ITC/3126/2005
Precisión	22/22=100%
Recall	22/22=100%

Tabla 8: Resultados obtenidos por el patrón Ordenes4

Podemos comprobar como los resultados son exactos. Si el usuario es capaz de introducir al patrón todo su conocimiento, éste devuelve unos resultados exactos, en lo que a la precisión y al recall se refiere.

A continuación procedemos a crear patrones destinados a reconocer entidades formadas por reales Decretos. Como en la anterior prueba ha quedado demostrado la conveniencia de una buena elección de ejemplos para obtener una buena eficiencia en los resultados del patrón generado, en este caso, vamos a escoger directamente ejemplos significativos. Para ello, y sobre el mismo texto, seleccionamos las siguientes entidades como ejemplos para la generación del patrón:

- 949/2001
- 1068/2007

Al lanzar la búsqueda sobre el patrón generado, este ha recuperado todos los decretos presentes en el texto, pero también ha identificados cadenas de texto que no se corresponden con una entidad.

Decretos1	
Precisión	20/68=29,4%
Recall	20/20=100%

Tabla 9: Resultados obtenidos por el patrón Decretos1

(Se omiten las entidades recuperadas por cuestiones de legibilidad).

Esto ha sido debido de nuevo a que no se ha especificado al patrón una característica que diferenciara a un decreto de, por ejemplo, una orden ministerial. En el siguiente ejemplo se ve perfectamente:

ITC/**2857/2008** es una orden ministerial, pero para el patrón, 2857/2008 podría pasar perfectamente como un decreto, pues es totalmente similar al segundo ejemplo introducido para la generación del patrón (1068/2007). Existen dos maneras de resolver este aspecto:

- Editar el patrón generado introduciendo como contexto la palabra "Decreto"
- Seleccionar como ejemplos los decretos escogidos anteriormente, con el espacio precedente incluido, para denotar que se trata de una palabra completa.

Si escogemos la segunda opción, los resultados obtenidos son inmejorables:

Decretos2	
949/2001	1804/2007
871/2007	1804/2007
326/2008	1804/2007
949/2001	949/2001
949/2001	949/2001
949/2001	949/2001
949/2001	949/2001
1434/2002	949/2001
949/2001	949/2001
1068/2007	949/2001
Precisión	20/20=100%
Recall	20/20=100%

Tabla 10: Resultados obtenidos por el patrón Decretos2

Realmente, y debido a que en los documentos BOE, un decreto siempre viene precedido de las palabras "Real Decreto", la mejor opción (aunque en este caso igual de eficiente) hubiera sido añadir al patrón generado inicialmente, el contexto "*Decreto*" con profundidad máxima =1. Si realizamos esta prueba, se obtienen los siguientes resultados que se muestran en la tabla 11:

Decretos1 con Contexto="*Decreto*"	
949/2001	1804/2007
871/2007	1804/2007
326/2008	1804/2007
949/2001	949/2001
949/2001	949/2001
949/2001	949/2001
949/2001	949/2001
1434/2002	949/2001
949/2001	949/2001
1068/2007	949/2001
Precisión	20/20=100%
Recall	20/20=100%

Tabla 11: Resultados obtenidos por el patrón Decretos1 (con Contexto)

A continuación se pasa a generar manualmente el patrón. Se generó el siguiente patrón:

S → A B C

A → {dígito}

B → "/200"

C → {digito}

Además se introdujeron las siguientes condiciones:

- A → Cardinalidad mínima = 3 y Cardinalidad mínima = 4
- Contexto → Debe aparecer la palabra "Decreto" con profundidad máxima igual a 1, es decir, al lado de la entidad

El patrón generado se denominó Decretos3. Los resultados obtenidos fueron muy buenos:

Decretos3	
949/2001	1804/2007
871/2007	1804/2007
326/2008	1804/2007
949/2001	949/2001
949/2001	949/2001
949/2001	949/2001
949/2001	949/2001

1434/2002	949/2001
949/2001	949/2001
1068/2007	949/2001
Precisión	20/20=100%
Recall	20/20=100%

Tabla 12: Resultados obtenidos por el patrón Decretos3

A continuación se realizaron pruebas sobre un tipo de entidad más variable y con mayor área de conocimiento: las fechas. Dado que en el BOE se da un gran número de apariciones de este tipo de entidad, el mismo corpus cargado era el idóneo para esta prueba. Primero se realizó un patrón manual, en el que intentaríamos transmitir al patrón todo el conocimiento necesario. Para la confección del patrón, lo primero que debíamos estudiar son las distintas maneras en las que se puede presentar una fecha: estas son las siguientes:

- La fecha contiene básicamente tres elementos: día, mes y año
- Los elementos anteriores están separados por:
  - Un espacio en blanco
  - La palabra "de"
  - Mes y año pueden ir separados por la palabra "del"
- El resto del conocimiento es el evidente para el lector (el mes tiene doce valores posibles, el día puede estar formado por uno o dos dígitos, etc.).

Este conocimiento puede ser adquirido también confeccionando un primer patrón y modificándolo de acuerdo a los resultados obtenidos. Con el fin de no sobrecargar la memoria en demasía, solo exponemos el patrón resultante final:

S → A B C D E F G H

A → {Separador}

B → {dígito}

C → " de " | {Separador}

D → "enero" | "febrero" | "marzo" | "abril" | "mayo" | "junio" | "julio" | "agosto" | "septiembre" | "octubre" | "noviembre" | "diciembre"

E → " de " | " del " | {separador}

F → "19" | "20"

G → {digito}

H → {digito}

Dado el gran número de entidades presentes en el texto, no se mostrarán las entidades capturadas.

Fechas1	
Precisión	51/51=100%
Recall	51/51=100%

Tabla 13: Resultados obtenidos por el patrón Fechas1

Los resultados obtenidos son fruto del éxito de trasmitir al patrón todo el conocimiento expuesto anteriormente, lo cual hace posible que el patrón no deje ninguna entidad sin capturar.

Para la confección del patrón generado automáticamente, es necesario transmitir el conocimiento citado en forma de ejemplos. Éstos han de ser lo más representativos posibles. A continuación se puede comprobar esta afirmación de manera muy evidente.

Para la confección de la primera versión del patrón, se escogieron los siguientes ejemplos:

- 3 de julio de 2007
- 1 de enero de 2007
- 31 diciembre 2008
- 1 de marzo de 2009
- 20 de abril de 2007
- 19 de febrero de 2002
- 28 de noviembre de 2003

Los resultados obtenidos fueron bastante aceptables:

Fechas2	
Precisión	44/46=95,6%
Recall	44/51=86,2%

Tabla 14: Resultados obtenidos por el patrón Fechas2

Por un lado, el patrón confeccionado, capturó por error las siguientes fechas:

- 0 de julio de 2007
- 0 de abril de 2007

Esta desviación es debida a que no se ha especificado al patrón que la fecha siempre aparece como una entidad separada, es decir, siempre va a venir precedida de un espacio en blanco, o un signo de puntuación. Para subsanar esta desviación basta con elegir los anteriores ejemplos con el separador (espacio en blanco o signo de puntuación) incluido con el que aparece en el texto.

Por otro lado, el patrón no capturó las siguientes entidades:

- 1 de enero del 2009
- 1 de enero de 2015
- 1 de octubre de 2009
- 1 de octubre de 2010
- 1 de enero de 2010
- 1 de diciembre de 2007
- 1 de enero de 2010

Veamos el por qué de estas exclusiones:

- No se ha especificado al patrón generado que el mes puede ser octubre.
- Tampoco se ha especificado al patrón que el mes y año pueden estar separados por la palabra "del".
- En los ejemplos introducidos, los años estaban comprendidos entre el 2002 y el 2009, por lo que el patrón no conoce que puede ser superior al año 2010.

Estas deficiencias se pueden subsanar añadiendo como ejemplos al patrón las fechas:

- 1 de enero del 2009
- 1 de enero de 2015
- 1 de octubre de 2009

Los resultados obtenidos fueron los siguientes:

Fechas3	
Precisión	49/49=100%
Recall	49/51=96,0%

Tabla 15: Resultados obtenidos por el patrón Fechas3

Podemos comprobar cómo los resultados mejoran ostensiblemente. Obviamente, si queremos extrapolar el uso de este patrón a cualquier tipo de texto, habría que introducir como mínimo una fecha relativa a cada posible mes, o en su defecto editar el patrón generado.

A continuación, y para demostrar que la herramienta funciona correctamente independientemente del corpus cargado, se procede a cargar en la aplicación el BOE del Ministerio de Economía y Hacienda del 17 de noviembre de 2008[BOE2], con el objetivo de recuperar las órdenes ministeriales presentes en este documento. Para recuperar estas entidades, simplemente es necesario modificar el patrón Ordenes4, y añadirle las siglas "EHA" y "HAC", que hacen referencia a las órdenes ministeriales de este ministerio. Los resultados obtenidos se muestran en la tabla 16:

Ordenes4 (modificado)	
EHA/3290/2008	EHA/702/2006
EHA/3895/2004	EHA/3398/2006
EHA/63/2005	HAC/1398/2003
EHA/1731/2005	EHA/3398/2006
EHA/1674/2006	HAC/3626/2003
EHA/3398/2006	EHA/3398/2006
HAC/1398/2003	HAC/3626/2003
EHA/3398/2006	HAC/3626/2003
EHA/3398/2006	HAC/3626/2003
HAC/1398/2003	HAC/3626/2003
Precisión	20/20=100%
Recall	20/20=100%

Tabla 16: Resultados obtenidos por el patrón Ordenes4 (modificado)

Se demuestra por tanto, que la herramienta funciona de manera independiente al corpus cargado. Vamos a probar también que la generación automática también funciona de manera independiente al corpus de prueba cargado.

Para ello, seleccionamos los siguientes ejemplos como entidades representativas de órdenes ministeriales:

- EHA/1674/2006
- EHA/3398/2005
- HAC/1398/2003
- EHA/63/2005

Los resultados obtenidos fueron también muy satisfactorios:

Ordenes5	
EHA/3290/2008	EHA/3398/2006
EHA/3895/2004	HAC/1398/2003
EHA/63/2005	EHA/3398/2006
EHA/1731/2005	HAC/3626/2003
EHA/1674/2006	EHA/3398/2006
EHA/3398/2006	HAC/3626/2003
HAC/1398/2003	HAC/3626/2003
EHA/3398/2006	HAC/3626/2003
EHA/3398/2006	HAC/3626/2003
HAC/1398/2003	
Precisión	19/19=100%
Recall	19/20=95%

Tabla 17: Resultados obtenidos por el patrón Ordenes5

Sólo una entidad no fue reconocida. Era aquella cuyo número presente entre ambas barras estaba formado por 3 dígitos. Esto sucede debido a que la generación automática está concebida de manera restrictiva. La herramienta sólo ha recibido como ejemplos válidos entidades compuestas por dos o cuatro dígitos entre las barras, y por ello descarta la entidad con tres dígitos. El por qué está concebida de esta manera es más comprensible de explicar desde el caso contrario. Se tiene por ejemplo las entidades que representan el identificador de, por ejemplo, los socios de un videoclub. Este identificador está compuesto de las iniciales del socio, seguido de un guión. A continuación aparecerá el año en el que el usuario se hizo socio, pero en caso de ser el actual, en vez del año se especificará el mes (del 01 al 12). Seguidamente otro guión y a continuación el DNI del usuario. Se exponen dos ejemplos:

- RMM-2006-47412357
- MML-06-67812462

En este caso, sería incorrecto que el patrón detectara cadenas con tres dígitos entre los guiones, a pesar de no haber recibido ningún ejemplo con este formato. Por ello, la generación automática se concibe de manera restrictiva, y es por ello necesario para asegurar su buen funcionamiento, el introducir ejemplos lo más significativos posibles de las variantes que pueden producirse. Obviamente a un patrón hay que expresarle todo el conocimiento, ya sea en forma de ejemplos o confeccionando el usuario las reglas.

Con los anteriores ejemplos, queda demostrado el buen funcionamiento de la herramienta, pero aún hay más aspectos que todavía no se han demostrado. Creamos sobre la ontología la clase "Prueba", donde albergaremos los patrones creados para esta causa. A continuación, creamos el patrón llamado Prueba1, que persigue probar el funcionamiento de las restricciones siguientes:

- Dos símbolos de la gramática han de tener el mismo contenido asociado
- Contexto
- Formato

El patrón es el siguiente:

S → A B C D E

A → {dígito}

B → {separador}

C → "UHI"

D → {separador}

E → {digito}

Además se introdujeron las siguientes condiciones:

- B y D han de tener el mismo contenido.
- C tiene que estar en formato subrayado.
- Contexto → Debe aparecer la palabra "identificador" con profundidad máxima igual a 1, es decir, al lado de la entidad.

Para probar el patrón creado, se precisa un archivo rtf, pues es necesario formato. Se utiliza parte de un archivo del BOE, introduciendo las siguientes palabras de manera dispersada:

- identificador 3 **UHI** 4
- identificador 5 **UHI** 1

Y las siguientes cadenas, que no ha de reconocer:

- identificador 3 UHI 4: la debe rechazar, pues el símbolo C (UHI) no está presentada en formato negrita.
- identificador 3-**UHI** 4: la debe rechazar, pues B ("-") y D (" ") no tienen el mismo contenido.
- matricula 3 **UHI** 4, Transferencias: la debe rechazar, pues no aparece en el contexto la palabra identificador.

Se expone el texto cargado, extraído del BOE del Ministerio de Economía y Hacienda del 17 de Noviembre de 2008, modificado para su adaptación a la prueba:

"carrera militar, por la legislacion autonomica correspondiente, por la normativa propia de cada universidad, por los identificador 3 ***UHI*** *4 convenios de adscripcion y por sus normas internas de organizacion y funcionamiento.Articulo 4. Ensenanzas de posgrado y lineas de investigacion.En los centros universitarios de la defensa se identificador 3 UHI 4 podran impartir tambien estudios conducentes a la obtencion de titulos oficiales de posgrado, en las modalidades de master y de doctor. Asimismo se definiran y desarrollaran lineas de investigacion que se consideren de interes para las Fuerzas Armadas y para la paz, la identificador 3* ***UHI****-4 seguridad y la defensa, colaborando, si procede, con otras entidades y organismos publicos de ensenanza e investigacion.Articulo 5. Financiacion.Los centros universitarios de la defensa dispondran de presupuesto propio financiado con cargo al capitulo matricula 3* ***UHI*** *4, Transferencias Corrientes, y al capitulo 7, Transferencias de Capital, del presupuesto del Ministerio de Defensa.Asimismo se financiaran con las subvenciones que, conforme a la legislacion vigente, se les puedan otorgar, los remanentes de identificador 5* ***UHI*** *1 tesoreria y cualesquiera otros ingresos percibidos en el ejercicio de sus actividades."*

Los resultados obtenidos fueron los esperados:

Prueba1	
3 **UHI** 4	5 **UHI** 1
Precisión	2/2=100%
Recall	2/2=100%

Tabla 18: Resultados obtenidos por el patrón Prueba1

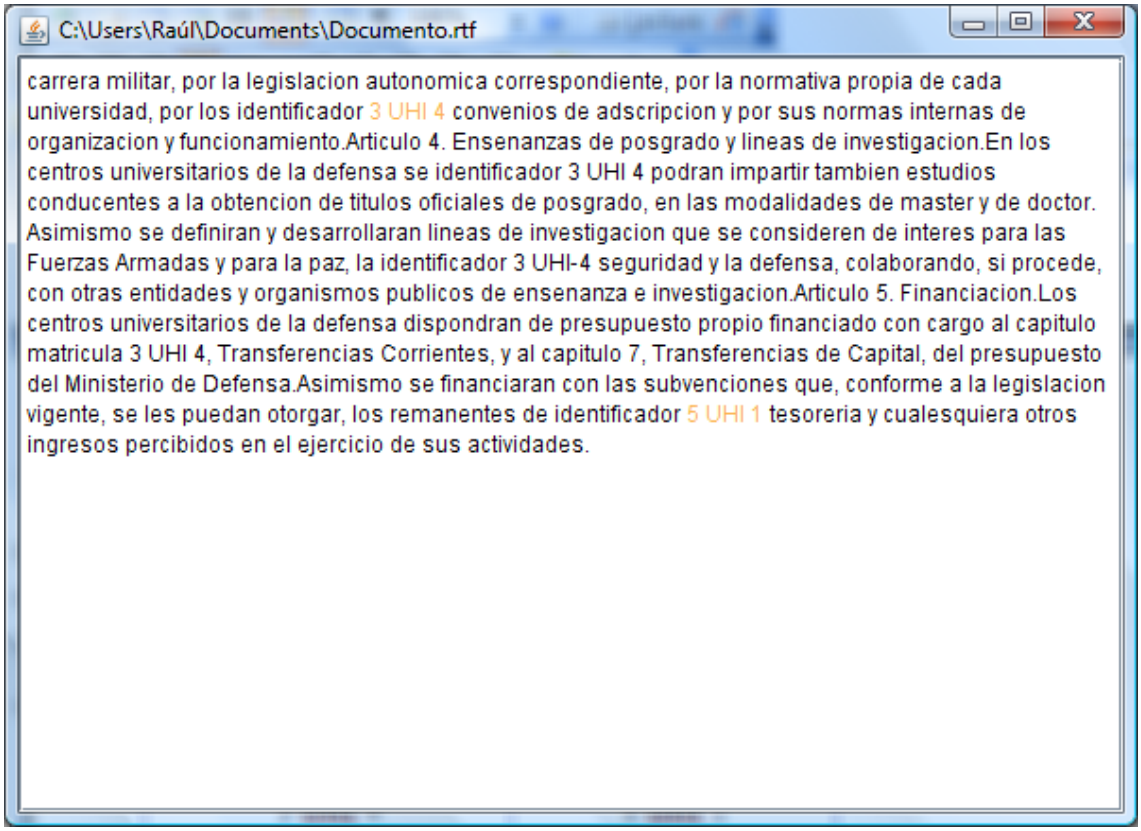


Ilustración 30: Resultados obtenidos para el patrón Prueba1

Se puede comprobar como no recuperó ninguna de las cadenas que se introdujeron para confundir al patrón, a causa de no cumplir todas las restricciones introducidas.

A continuación se realizó otra prueba, con el objetivo de probar otras restricciones:

- Dos símbolos de la gramática han de tener la misma cardinalidad asociada
- Excepción asociada a un elemento de la gramática

El patrón es el siguiente:

S → A B C D

A → {dígito}

B → "/"

C → Rango Numérico{1,9}

D → {caracter}

Además se introdujeron las siguientes condiciones:

- B y D han de tener la misma cardinalidad.
- B y D tienen cardinalidad mínima 1 y máxima 4.
- A no puede tener el valor 2009.

Para probar el patrón creado, se creó un corpus de prueba similar al anterior, en el que se introdujeron las siguientes entidades:

- 33/98a
- 143/368t
- 2008/1342f

Y las siguientes cadenas, que no ha de reconocer:

- 45/567g: la debe descartar pues A (45) y C (567) no tienen la misma cardinalidad.
- 2009/3456h: la debe rechazar puesto que A (2009) cumple la excepción.
- 34/20j: la debe rechazar, puesto que C (20), uno de los dígitos no cumple el rango numérico (1,9).

Se expone el texto cargado:

"*carrera militar, por la legislacion autonomica correspondiente, por la normativa propia de cada universidad, por los convenios de adscripcion y por sus normas internas de organizacion 33/98a y funcionamiento.Articulo 4. Ensenanzas de posgrado y lineas de investigacion.En los centros universitarios de la defensa se identificador podran impartir tambien estudios conducentes a la obtencion de titulos oficiales de posgrado, en las modalidades de master y de doctor. Asimismo 45/567g se definiran y desarrollaran lineas de investigacion que se consideren de interes para las Fuerzas Armadas y para la paz, la seguridad y la defensa, colaborando, si procede, con otras entidades y organismos publicos de ensenanza e investigacion.Articulo 5. Financiacion.Los centros universitarios de la defensa 143/368t dispondran de presupuesto propio financiado con cargo al capitulo, Transferencias 2008/1342f Corrientes, y al capitulo 7, Transferencias de Capital, del presupuesto del Ministerio de Defensa.Asimismo se financiaran con las subvenciones que, conforme a la legislacion vigente, se les puedan otorgar, los remanentes 2009/3456h de tesoreria y cualesquiera otros ingresos percibidos en el ejercicio de sus actividades*"

Los resultados obtenidos fueron también los esperados:

Prueba2	
33/98a	143/368t
2008/1342f	5 **UHI** 1
Precisión	3/3=100%
Recall	3/3=100%

Tabla 19: Resultados obtenidos por el patrón Prueba2

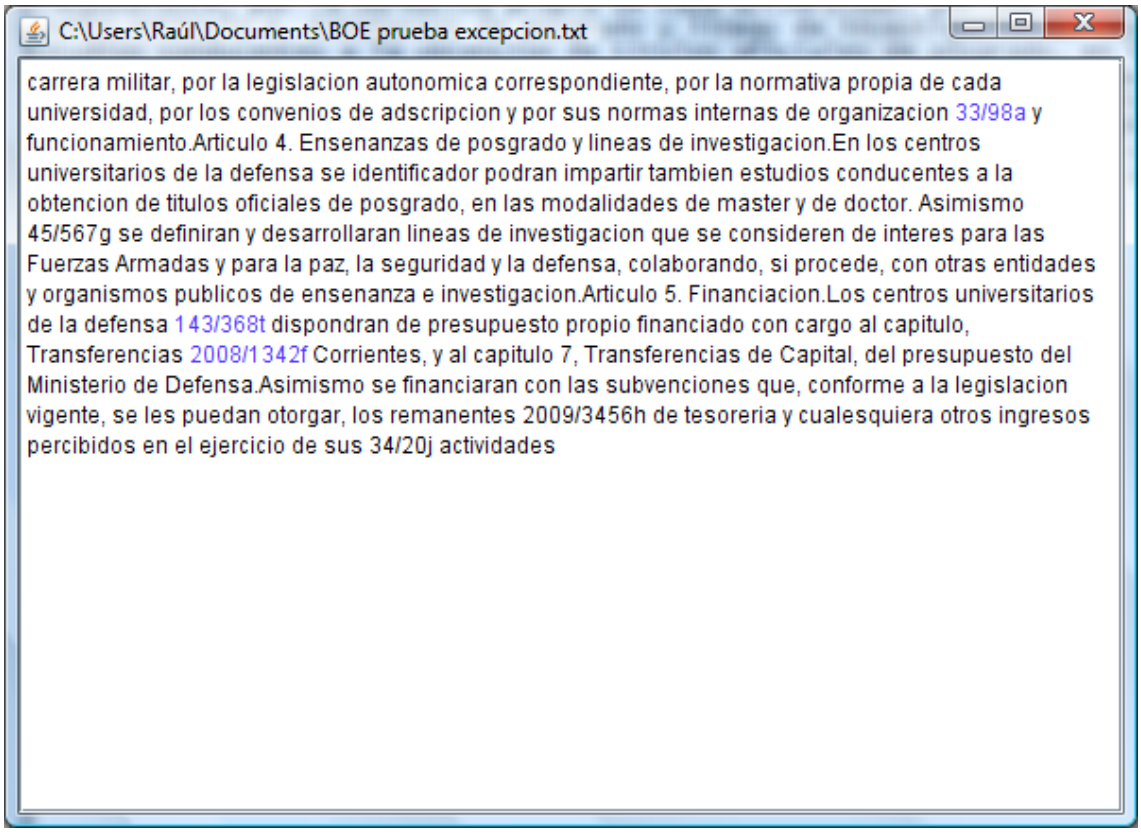


Ilustración 31: Resultados obtenidos para el patrón Prueba2

En este caso el patrón tampoco recuperó ninguna de las cadenas de texto que cumplían todas las restricciones introducidas a excepción de una, mientras que recuperó todas las entidades presentes en el texto.

6 CONCLUSIONES

Las herramientas que permiten una identificación y reconocimiento de entidades tienen un gran potencial para la extracción de información. En el presente proyecto se han analizado varias de estas herramientas. La selección se hizo en función de su accesibilidad (mayoritariamente gratuitas) y su popularidad.

Tras un análisis de las herramientas se detectaron deficiencias relativas a:

- Su usabilidad
- Su capacidad para identificar y mostrar nuevos patrones

Para solventar estos problemas se ha diseñado una aplicación que muestra mejoras en ambos apartados.

Usabilidad:

- El almacenamiento de los patrones de las entidades en forma de ontología presenta un mayor grado de reuso futuro.
- Interfaz de resultados intuitiva. La visualización se ha mejorado mediante la utilización de distintos colores para cada tipo de entidad. Además se muestran las entidades en forma de árbol jerárquico, mejorando su interpretación.
- La falta de usabilidad asociada a las ontologías (debido a la dificultad para el usuario inexperto del lenguaje empleado) se solucionó con un editor para su gestión.

Identificación:

- Se han habilitado dos formas de identificación de patrones: Generación directa mediante una gramática propuesta por el usuario; y Generación automática a partir de ejemplos presentes en un texto y marcados por el usuario.
- La definición mediante patrones de las entidades dota de mayor flexibilidad a la definición de patrones de entidades. frente a las habituales expresiones regulares, o incluso frente al aprendizaje automático.

Por último las pruebas efectuadas en términos de precisión (oscilación entre 95-100%) y recall (entre 86.2-100%) muestran buenos ratios de recuperación. Las pruebas se realizaron con varios textos de distinta temática procedentes del BOE. Los resultados muestran cómo los corpus no inciden de forma significativa en las

ratios. Los resultados fueron buenos aún en los casos en los que se insertaban términos para confundir a los patrones.

En definitiva se ha desarrollado una aplicación que mejora a las estudiadas y permite una evolución futura prometedora.

7 TRABAJOS FUTUROS

A continuación se enumeran una serie de posibles trabajos futuros de cara a ampliar la potencia y funcionalidad de la herramienta.

- Aplicación Web: al estar diseñada la herramienta en código Java y usar ontologías OWL para el almacenamiento de patrones, la herramienta sería fácilmente transformable a una aplicación Web (applet) y, paralelamente, podría importar ontologías Web (introduciendo su URL) y trabajar sobre documentos presentes en la Web. Esto supondría aumentar la potencia de la herramienta y admitir páginas HTML como entrada de texto.
- Aceptación de listados: como se ha podido ver durante el estado del arte, muchas herramientas trabajan con listados para capturar entidades no basadas en la morfología de la palabra, como puedan ser los nombres de las ciudades. Para detectar de una manera eficiente este tipo de entidades, la única forma hasta ahora probada es mediante listados. La herramienta podría recibir el listado, y a partir de él generar un patrón muy sencillo, consistente en una gramática cuya única regla deriva en una serie de literales, cada uno de ellos se corresponde con cada entidad incluida en el listado.
- Generación Automática a partir de listados: sería interesante poder ampliar la entrada de la generación automática de patrones a los listados. La herramienta recibiría un listado de entidades, y a partir de ellos generaría el patrón correspondiente, de tal manera que sería posible editar el listado para regenerar el patrón. De esta manera, se mejoraría la usabilidad de la herramienta.
- Potencia de la generación automática: se podría dotar a la generación automática de patrones de mayor potencia, pues en el presente proyecto han quedado sin utilizar varios elementos proporcionados por los patrones, tales como el contexto, el formato o las comprobaciones horizontales (igual cardinalidad y/o contenido entre dos trozos de una misma entidad). Se dotaría al patrón generado de mayor potencial y eficacia.
- Patrones anidados: dentro de una clase de la ontología, pueden existir patrones que guardan una relación "todo-parte". Es decir, las entidades que captura uno están incluidas en las entidades que captura el otro. Un ejemplo sencillo sería el patrón reconocedor de direcciones postales y el reconocedor de códigos postales. La herramienta podría soportar esta relación, mostrando los patrones de forma jerárquica.

- Evaluación de la usabilidad mediante estudios de usuarios.
- Aumentar el número y tipología de corpus para evaluar el comportamiento de la aplicación.

8 BIBLIOGRAFÍA

[ANN1] ANNIE. A Nearly-New Information Extraction System. http://gate.ac.uk/sale/tao/index.html#annie. Última visita: Enero de 2009

[BOE1] Boletín Oficial del Estado del Ministerio de Industria y Comercio del 31 de Diciembre de 2008. http://www.boe.es/boe/dias/2008/12/31/pdfs/A52686-52700.pdf. Última visita: Enero de 2009

[BOE2] Boletín Oficial del Estado del Ministerio de Ecnomía y Hacienda del 17 de Noviembre de 2008. http://www.boe.es/boe/dias/2008/11/17/pdfs/A45583-45621.pdf. Última visita: Enero de 2009

[GAT1] GATE. A General Architecture for Text Engineering. http://gate.ac.uk. Última visita: Enero de 2009

[GRAM1] Rudolf Ortega I Robert. Lenguajes naturales y lenguajes formales. Diciembre 1993. ISBN 84-477-0264-2.

[GRAM2] Tao Jiang, Ming Li, Bala Ravikumar, Kenneth W. Regan, formal Grammars and Languajes. Capítulo 25 del libro Handbook on Algorithms and Theory of Computation. Mikhail J. Atallah. November 1998. ISBN: 0849326494

[GRAM3] Chomsky, N. Formal properties of grammars. 1963. Del libro Mathematical Psychology Vol. 2, 323-418.

[JEN1] Jena – A Semantic Web Framework for Java. http://jena.sourceforge.net. Última visita: Enero de 2009

[JEN 2] Ryan Lee. Scalability Report on Triple Store Applications. http://simile.mit.edu/reports/stores/index.html. Última visita: Enero de 2009

[KIM1] KIM. The KIM Platform: Knowledge & Information Management. http://www.ontotext.com/kim/index.html. Última visita: Enero de 2009

[KIM2] B. Popov, A. Kiryakov, A. Kirilov, D. Manov, D. Ognyanoff, M. Goranov. KIM, Semantic Annotation Platform. Proceedings of Second International Semantic Web Conference, ISWC 2003.

[LUC1] Lucene. The Apache Lucene Project. http://lucene.apache.org. Última visita: Enero de 2009

[MNM1] MnM. Ontology Driven Semi-Automatic and Automatic Support for Semantic Web. http://projects.kmi.open.ac.uk/akt/MnM. Última visita: Enero de 2009

[MNM2] M. Vargas-Vera, E. Motta, J. Domingue, M. Lanzoni, F. Stutt, F. Ciravegna. MnM: Ontology Driven Semi-Automatic and Automatic Support for Semantic Markup. In 13th International Conference on Knowledge Engineering and Management (EKAW 2002).

[ONT1] OntoText Gazetteer. http://gate.ac.uk/sale/tao/index.html#x1-2410009.4 . Última visita: Enero de 2009

[ONT2] OntoText Lab. Sitio oficial en Internet http://www.ontotext.com. Última visita: Enero de 2009

[OWL1] Ivan Herman Web Ontology Language (OWL). W3C Semantic Web Activity. http://www.w3.org/2004/OWL/. Última visita: Enero de 2009

[PRO1] The Protegé Ontology Editor and Knowledge Adquisition System. http://protege.stanford.edu. Última visita: Enero de 2009

[PRO2] Carlos Carrascosa Casamayor. Análisis de Protegé-2000. http://personales.upv.es/ccarrasc/doc/2001-2002/Protege2000/PROTEGE.htm. Última visita: Enero de 2009

[SES1] Sesame. Cornerstone of the Semantic Web. http://www.aduna-software.com/technologies/sesame/overview.view. Última visita: Enero de 2009

[SOF1] SOFA, Simple Ontology Framework API. http://sofa.projects.semwebcentral.org Última visita: Enero de 2009

[WEB1] Luis Sánchez Fernández, Norberto Fernández-García. La Web Semántica: fundamentos y breve "estado del arte". Novática. Diciembre 2005. ISSN 0211-2124. Enlace: http://www.ati.es/novatica/2005/178/178-6.pdf. Última visita: Enero de 2009