UNIVERSIDAD AUTÓNOMA METROPOLITANA

UNIDAD AZCAPOTZALCO

DIVISIÓN DE CIENCIAS BÁSICAS E INGENIERÍA

Sistema de almacenamiento semántico y recuperación de textos de investigación mediante ontologías

REPORTE FINAL DE PROYECTO TERMINAL

Alumno

Fernando Tébar Martínez

2123999433

Asesora:

Dra. Maricela Claudia Bravo Contreras

Departamento de Sistemas

MAYO 2013

Contenido

Introducción

Después de comprobar la importancia y también la gran dificultad que entraña el análisis del lenguaje natural en un texto de investigación, nuestro objetivo ha sido implementar una herramienta que garantizara un almacenamiento persistente de toda la información obtenida tras el reconocimiento de los textos en una base de datos y una ontología.

Esta tarea conlleva la utilización de diferentes recursos y herramientas disponibles en la web. En general no hay mucha información sobre ontologías, más que la que proporcionan los sitios web de Protegé [1], Jess [2] y algún otro recurso usado. A pesar de ello, con la información disponible online y ejemplos anteriores de ontologías se ha podido desarrollar satisfactoriamente la totalidad del proyecto que se describió en la propuesta presentada en noviembre del año pasado.

El proyecto de almacenamiento semántico constituye una extensión al proyecto previo sobre el que trabajamos y trata de dotar a sus resultados de una persistencia en el tiempo más allá de la inmediatez de la búsqueda. Es imprescindible conocer en profundidad los pormenores del diseño de ontologías para poder dotar de importancia a todos los campos recuperados en el análisis.

Para poder transformar este almacenamiento semántico en una herramienta de búsqueda útil, es decir, fiable y rápida, de búsqueda, se usa un mecanismo fundamental basado en la ontología y las reglas de inferencia. Para la parte final del buscador, las reglas de inferencia proporcionan un potente recurso de búsqueda en los individuos registrados en la ontología.

Revisión de objetivos específicos

- ✓ Adecuar el resultado del procesamiento de textos de tal manera que reconozca los atributos importantes en la ontología.

SRIS (Sistema de Recuperación de Información Semántico) entregaba unos datos poco fiables después de parsear y extraer los campos de cada uno de los artículos, sin embargo, se comprobaba que esa información podría valer para cada uno de los campos. Si alguno de los campos más importantes faltara, no se incluiría ese documento en la ontología, de manera que las instancias en la ontología corresponden a documentos cuyos datos estaban correctos en su forma.

- ✓ Sincronizar la captación de los elementos con el almacenamiento inmediato en la ontología y base de datos.

La información recuperada por SRIS llega a la clase *Módulo* y allí mismo se inserta en la ontología y posteriormente se crean instancias de las clases *Creación* e *Inserción*, en las que primero se crea la base de datos y las tablas y después se insertarán los datos que se han insertado en la ontología.

- Relacionar todos los datos útiles del texto con alguna instancia en la ontología, y por tanto con alguna tabla en la base de datos.

El módulo comprueba que los campos importantes en cada documento leído están completos y que se guarda una consistencia semántica entre ellos. Es decir, si alguno de los campos está incompleto o no existe, se descarta la inserción de ese documento en la ontología y en la base de datos.

- Completar todas las instancias de la ontología con la información de cada texto.

Como se ha comentado previamente, si todos los campos del documento son válidos léxica y semánticamente se insertarán en la ontología, si no es así, se descartará todo el documento para que posteriormente las búsquedas no sean inútiles.

- Descartar la información redundante en los textos y duplicada en la base de datos.

En caso de que se quiera insertar un documento con cualquier campo ya existente en la ontología, ésta lo actualizará sin duplicarlo. Por ejemplo, si una universidad ya existe y se quiere insertar un autor que trabaja allí, la instancia de esa universidad añadirá un campo *works* para el autor nuevo. Lo mismo ocurrirá en la base de datos. Añadiendo las claves primarias y únicas evitamos el problema de la repetición, sin embargo, teniendo directorios podemos garantizar la existencia de cualquier autor con varios artículos o cualquier universidad con varios autores.

Desarrollo del proyecto

Diseño

Código Java

El diseño del proyecto queda inevitablemente acotado y determinado por el proyecto anterior ya que constituye una extensión del mismo, desarrollando nuevas funcionalidades y mejorando otras que ya existían. El sistema de recuperación de información semántico (SRIS) ha sido desarrollado por otra alumna de la UAM-A [3] y consiste en un módulo software que lee los textos de investigación y recupera la información importante: autor, nombre de la publicación, universidad, año, país, *keywords* y el *abstract* del documento.

El nuevo proyecto, descrito en este documento, es otro módulo software, que es una ampliación del proyecto original. Se ha respetado la estructura del código SRIS y se ha integrado el nuevo módulo diseñado y desarrollado en este proyecto junto con otros módulos que gestionan la interfaz, recursos de procesamiento con GATE [4] y otras clases usadas.

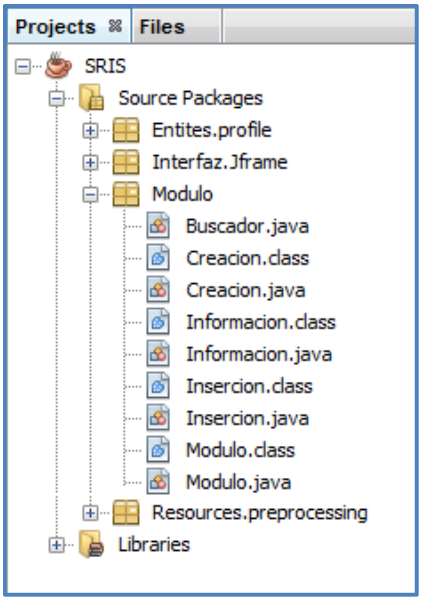


Figura1: Paquetes en NetBeans

A partir de esta información, en muchos casos incompleta, que se obtenía mediante la ejecución de SRIS, se representaban estos datos mediante estructuras en memoria dinámica, después se insertaban en la ontología y posteriormente se incorporaban a una base de datos.

El operativa del diseño de las clases se hace siguiendo la siguiente lógica:

1. La clase de SRIS *AnnotationGroup* lee los documentos uno a uno y obtiene la información que se va a usar por la clase *Modulo* que los recibe.

2. La clase *Modulo* se encarga de recibir la información enviada por a clase de SRIS *AnnotationGroup* e inserta los datos en un *template* de la ontología creada con *Protegé*.

3. A continuación, desde la clase *Modulo* se llama al constructor de la clase *Creación*,

4. El constructor se encarga conectar con el gestor de bases de datos MySQL y crea 4 tablas en una base de datos creada previamente.

5. Después, se crea la clase *Inserción*, que ejecuta las inserciones de los datos respetando la misma estructura definida para la ontologia pero en código MySQL desde la clase Java.

En otra secuencia de ejecución, se ha diseñado y desarrollado la clase *Buscador*, que implementa la lectura de la ontología y las reglas de inferencia para la parte del buscador que está en la interfaz de la aplicación.

Se muestra el diagrama UML [5] hecho con la herramienta Dia [6] junto con el resto de módulos de los que recogen información y la almacenan. El código desarrollado está en el cuadro grande con líneas punteadas y discontinuas, y los módulos se sitúan fuera.

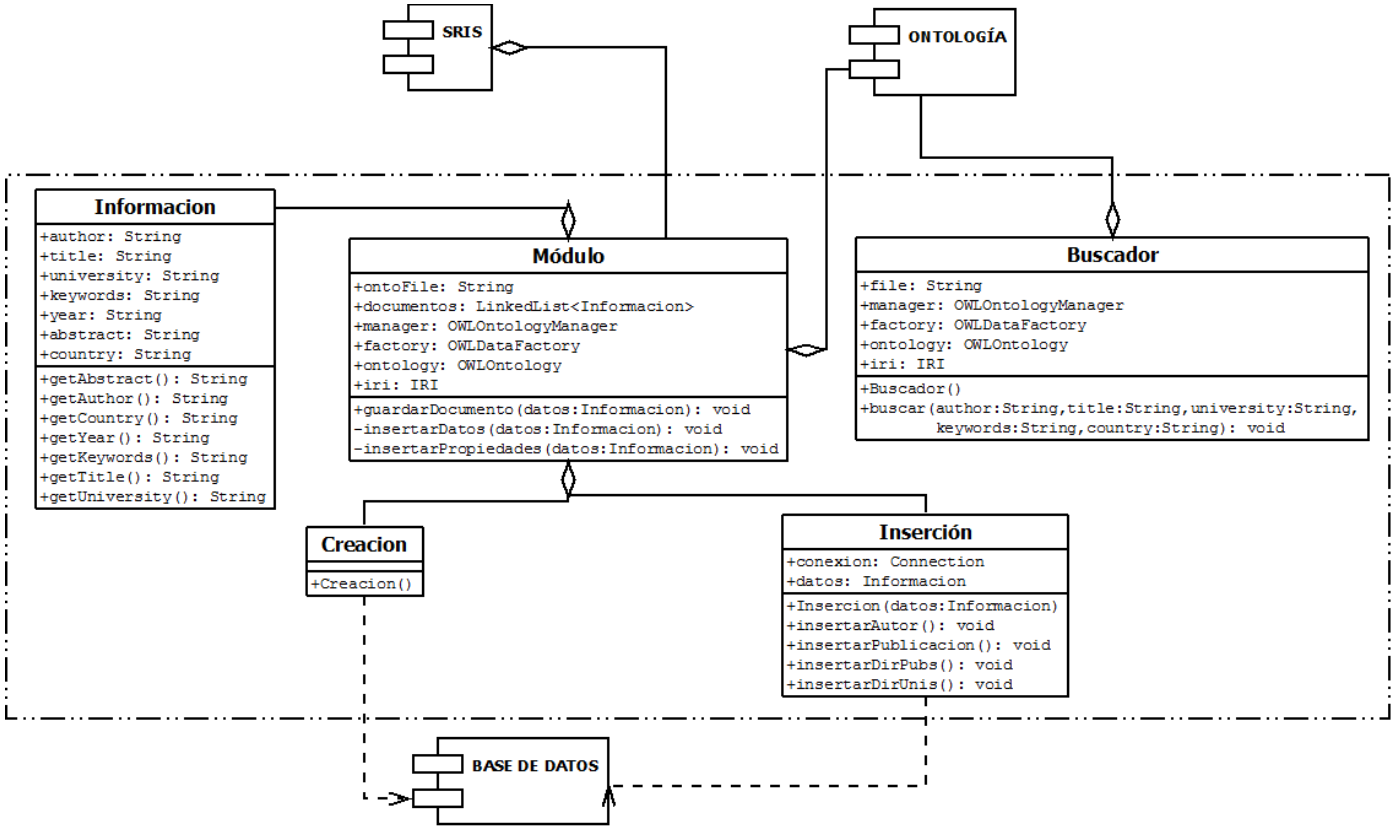


Diagrama UML

Implementación

JAVA

La clase principal, que recibe la información necesaria del SRIS es la clase *Módulo*, a continuación se detalla un fragmento de la inserción de estos campos en la ontología:

```
String aut =info.getAuthor();

String univ = info.getUniversity();

String publi =info.getTitle();

publi = publi.replace('\n', ' ');

publi= publi.replace(' ', '_');

aut = aut.replace('\n', ' ');

aut= aut.replace(' ', '_');


// Creamos autor

OWLNamedIndividual autor =
factory.getOWLNamedIndividual(IRI.create(iri.toString() + "#"+ aut));

OWLClass investigador = factory.getOWLClass(IRI.create(iri.toString()+
"#Author"));

OWLClassAssertionAxiom classAssertion = factory.getOWLClassAssertionAxiom
                                                    (investigador, autor);

man.addAxiom( ontology, classAssertion);


//Creamos universidad

univ= univ.replace(' ', '_');

OWLNamedIndividual uni = factory.getOWLNamedIndividual
                        (IRI.create(iri.toString()  + "#"+univ));

OWLClass universidad = factory.getOWLClass
                        (IRI.create(iri.toString() + "#University")  );


OWLClassAssertionAxiom classAssertion2 = factory.getOWLClassAssertionAxiom
                                          (universidad, uni);

man.addAxiom( ontology, classAssertion2);
```

```
// Anadimos la relacion "trabaja"

OWLObjectPropertyExpression owlObjecttype = factory.getOWLObjectProperty
(IRI.create(iri.toString() + "#works"));

OWLObjectPropertyAssertionAxiom dataProperty
=factory.getOWLObjectPropertyAssertionAxiom(owlObjecttype, autor, uni);

man.addAxiom( ontology, dataProperty);

OWLObjectPropertyExpression owlObjecttype2 = factory.getOWLObjectProperty
(IRI.create(iri.toString() + "#hosts"));


OWLObjectPropertyAssertionAxiom dataProperty2 =
factory.getOWLObjectPropertyAssertionAxiom(owlObjecttype2, uni, autor);

man.addAxiom( ontology, dataProperty2);


// Anadimos la relacion "publica"

OWLNamedIndividual publicacion = factory.getOWLNamedIndividual
                       (IRI.create(iri.toString()  +"#"+publi));

OWLClass publica = factory.getOWLClass(IRI.create(iri.toString()
                                    + "#Publication"));

OWLClassAssertionAxiom classAssertionn = factory.getOWLClassAssertionAxiom
                                         (publica, publicacion);

man.addAxiom( ontology, classAssertionn);


OWLObjectPropertyExpression owlObjecttype3= factory.getOWLObjectProperty
                              (IRI.create(iri.toString()+"#publishes"));

OWLObjectPropertyAssertionAxiom dataProperty3 =
factory.getOWLObjectPropertyAssertionAxiom(owlObjecttype3, autor,
publicacion);

man.addAxiom( ontology, dataProperty3);

OWLObjectPropertyExpression owlObjecttype4 = factory.getOWLObjectProperty
                              (IRI.create(iri.toString()+"#isPublishedBy"));

OWLObjectPropertyAssertionAxiom dataProperty4 =
factory.getOWLObjectPropertyAssertionAxiom(owlObjecttype4, publicacion,
autor);

man.addAxiom( ontology, dataProperty4);

insertarPropiedades(info);
```

```
try {
      man.saveOntology(ontology);
} catch (OWLOntologyStorageException e) {
      e.printStackTrace();
}
Insercion insertar = new Insercion(info);      }}
```

SWRL

SWRL (Semantic Web Rule Language) [7] se ha convertido en el lenguaje de reglas estándar de la Web semántica. La especificación se presentó en 2004 al W3C por la Agencia Nacional de desarrollo de Canadá y la Universidad de Stanford. Proporciona la capacidad reglas "tipo cuerno" expresadas en términos de relaciones con sintaxis OWL. Así como otros lenguajes, las reglas de SWRL están escritas con estructura de antecedente y consecuente. En la terminología de SWRL, el antecedente constituye el cuerpo de la regla y el consecuente como la cabeza. La cabeza y el cuerpo consisten en la suma de uno o más átomos.

Ontología

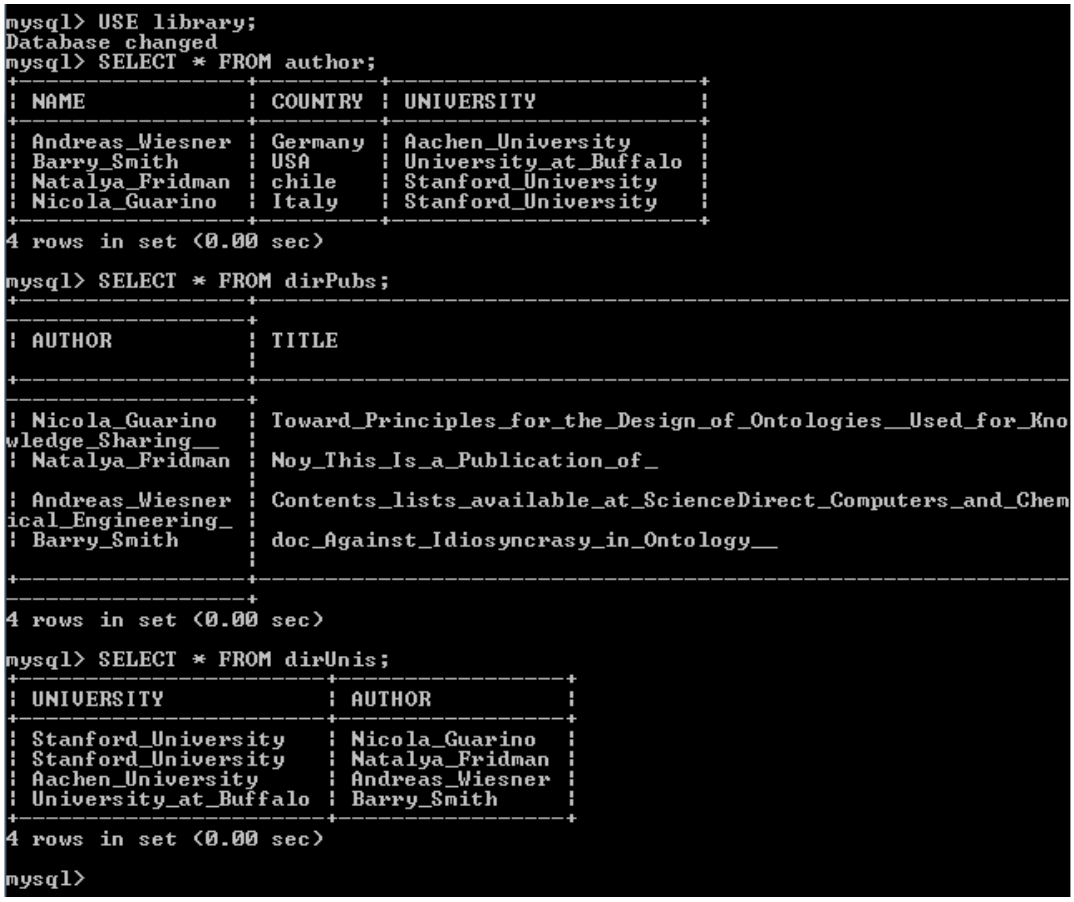
Este es un ejemplo de los tres tipos de clases que tenemos en la ontología. En color morado se indica el tipo. En color rojo se encuentra la universidad, que está en el nombre de su instancia y en la propiedad del autor que trabaja allí. En azul el nombre del autor, que figura en su instancia, en la universidad donde trabaja y en el artículo que ha publicado. En verde está el nombre de la publicación y la referencia en la instancia del autor. El código está escrito en lenguaje de ontologías OWL [8].

```
<owl:NamedIndividual rdf:about="http://www.owl-ontologies.com/Ont
/Ontology.owl#Aachen_University">
        <rdf:type rdf:resource="http://www.owl-ontologies.com/Ont
/Ontology.owl#University"/>
        <hosts rdf:resource="http://www.owl-ontologies.com/Ont
/Ontology.owl#Andreas_Wiesner"/>
</owl:NamedIndividual>
```

```
<owl:NamedIndividual rdf:about="http://www.owl-ontologies.com/Ont
/Ontology.owl#Andreas_Wiesner">

        <rdf:type rdf:resource="http://www.owl-ontologies.com/Ont
/Ontology.owl#Author"/>

        <Country rdf:datatype=http://www.w3.org/2001/XMLSchema#string
>Germany</Country>

        <works rdf:resource="http://www.owl-ontologies.com/Ont
/Ontology.owl#Aachen_University"/>

        <publishes rdf:resource="http://www.owl-ontologies.com/Ont
/Ontology.owl#Contents_lists_available_at_ScienceDirect_Computers_and_
Chemical_Engineering_"/>

</owl:NamedIndividual>


    <owl:NamedIndividual rdf:about="http://www.owl-ontologies.com/Ont
/Ontology.owl#Contents_lists_available_at_ScienceDirect_Computers_and_
Chemical_Engineering_">

        <rdf:type rdf:resource="http://www.owl-ontologies.com/Ont
/Ontology.owl#Publication"/>

        <Abstract rdf:datatype=http://www.w3.org/2001/XMLSchema#string
></Abstract>

        <Keywords rdf:datatype=http://www.w3.org/2001/XMLSchema#string
>Keywords:Ontology</Keywords>

        <isPublishedBy rdf:resource="http://www.owl-ontologies.com/Ont
/Ontology.owl#Andreas_Wiesner"/>

    </owl:NamedIndividual>
```

Base de datos

Aquí se muestra una captura de la base de datos MySQL [9] después de ser actualizada:



```
mysql> USE library;
Database changed
mysql> SELECT * FROM author;
+-----------------+---------+-----------------------+
| NAME            | COUNTRY | UNIVERSITY            |
+-----------------+---------+-----------------------+
| Andreas_Wiesner | Germany | Aachen_University     |
| Barry_Smith     | USA     | University_at_Buffalo |
| Natalya_Fridman | chile   | Stanford_University   |
| Nicola_Guarino  | Italy   | Stanford_University   |
+-----------------+---------+-----------------------+
4 rows in set (0.00 sec)

mysql> SELECT * FROM dirPubs;
+-----------------+-------------------------------------------------------------------
-------------------+
| AUTHOR          | TITLE
                  |
+-----------------+-------------------------------------------------------------------
-------------------+
| Nicola_Guarino  | Toward_Principles_for_the_Design_of_Ontologies__Used_for_Kno
wledge_Sharing__  |
| Natalya_Fridman | Noy_This_Is_a_Publication_of_
                  |
| Andreas_Wiesner | Contents_lists_available_at_ScienceDirect_Computers_and_Chem
ical_Engineering_ |
| Barry_Smith     | doc_Against_Idiosyncrasy_in_Ontology__
                  |
+-----------------+-------------------------------------------------------------------
-------------------+
4 rows in set (0.00 sec)

mysql> SELECT * FROM dirUnis;
+-----------------------+-----------------+
| UNIVERSITY            | AUTHOR          |
+-----------------------+-----------------+
| Stanford_University   | Nicola_Guarino  |
| Stanford_University   | Natalya_Fridman |
| Aachen_University     | Andreas_Wiesner |
| University_at_Buffalo | Barry_Smith     |
+-----------------------+-----------------+
4 rows in set (0.00 sec)

mysql>
```

Herramientas

NetBeans: El proyecto se implementó utilizando java como lenguaje de programación, mediante el uso de NetBeans 7.2.1 [10] como entorno de desarrollo integrado (IDE). El uso de este IDE permite al programador un desarrollo más ágil y una mejor estructuración del proyecto. Esta herramienta permitió una mejor documentación del código fuente de la aplicación así como la realización de las pruebas unitarias.

Mediante el uso de la opción *debug* se pudieron encontrar errores sobre el funcionamiento del sistema, ya que esta opción te permite marcar los puntos de interrupción en donde se tuviera algún funcionamiento inesperado del sistema. Debido a que este tipo de errores es complicado encontrar donde existe un error, con el debug se pudo seguir el cambio en las variables y ver donde se tenía un valor que no era el esperado y corregir la lógica de programación.

Protégé: Se utilizó Protégé 3.4.8 como editor de ontologías, esta herramienta facilita la creación de los modelos ontológicos, el entorno gráfico del editor es intuitivo y tiene varias pestañas para trabajar con alguna sección específica que conforma la ontología, como puede ser las clases que conforman el modelo, las propiedades de las clases y las propiedades de tipos de datos, entre otros.

Permite agregar o eliminar clases del modelo de una forma intuitiva, también permite la creación de las propiedades de objetos y propiedades de datos, establecer las relaciones que existen entre las clases, agregar que propiedades tendrá cada objeto y definir el rango y el dominio que existe en cada relación que se defina entre las clases.

Protegé principalmente me sirvió para familiarizarme con las ontologías, propiedades del objeto, propiedades de los datos, etc. También fue fundamental al principio para poder crear el template sobre el cual mi programa iba a insertar toda la información.

APIs

OWL API: Esta API proporciona clases y métodos para el trabajo con ontologías, permitiendo la creación, manipulación y lectura. Particularmente esta API es muy importante en el proyecto ya que nos permite realizar el poblado de la ontología, después de haber cargado el template del modelo ontológico. Esta API esta implementada en java lo que permitió una fácil integración con el proyecto, de tal manera que la integración de ambas resulto sencilla y no se produjo ningún conflicto al trabajar con ella. OWL API [11] cumplió con el objetivo de crear una colección de servicios web con los datos obtenidos de los archivos de descripción, permitiendo establecer las relaciones definidas en el modelo ontológico.

JESS: Jess es un motor de reglas y entorno de programación desarrollado completamente en Java. Jess permite construir software Java con capacidad de razonamiento proporcionando conocimiento en forma de reglas declarativas. Debido a su tamaño y ligereza es uno de los motores de reglas más rápidos que existen.

Jess utilizado una versión mejorada del algoritmo de Rete para procesar estas reglas. El algoritmo de Rete es un mecanismo muy eficiente para encontrar soluciones a problemas con muchas combinaciones. También tiene muchas capacidades únicas, incluyendo encadenamiento hacia atrás además de poder razonar sobre objetos Java.

Utilizamos esta herramienta en el buscador para crear el motor de inferencia y así ejecutar las reglas sobre la ontología. Usamos el api para crear el modelo de la ontología y después crear el motor.

Pruebas

Para probar la fiabilidad del sistema SRIS sobre el que se fundamenta mi trabajo, ejecutamos la recuperación de: nombre de un autor, nombre del artículo de investigación, el *abstract*, la universidad en donde trabaja el autor y su país. Ejecutamos el proceso de extracción en 38 artículos diferentes de muy diversos formatos.

Para medir campo por campo si el resultado es correcto, usamos el siguiente criterio.

- Si el nombre del autor o el de la publicación es correcto, se dará 1 punto.
- Si es incompleto o añade alguna otra palabra, se le puntúa con 0'5 puntos.
- Si no lo reconoce serán 0 puntos.

Presentamos la siguiente tabla donde se muestra el cómputo de las lecturas de cada campo.

	Veces completo	Veces incompleto	Veces erróneo	Total	% (Total/38)
Autor	22	5	11	24.5	**64.4**
Publicación	10	7	21	13.5	**35.5**
Universidad	18	3	17	19.5	**51.3**
Abstract	17	3	18	18.5	**48.6**
País	19	0	19	19	**50.0**

Teniendo en cuenta que en total son 190 lecturas de campo (38 archivos * 5 campos), calculamos el porcentaje de campos leídos correctamente: 86/190 = 45.2 %. Campos leídos parcialmente: 18/190 = 9%. Campos leídos erróneamente: 86 / 190 = 45.2 %.

Esto arroja dos resultados:

- Tenemos la misma probabilidad de que el sistema reconozca bien el campo como que no lo reconozca.
- Si sumamos a los correctos, los incompletos (0.5 puntos), tenemos 95 aciertos/190 = 50 % de que reconozca el campo completa o parcialmente.

Hasta aquí se ha calculado la fiabilidad de la herramienta por campo analizado, ahora presentamos los resultados obtenidos por documento en un total de 38.

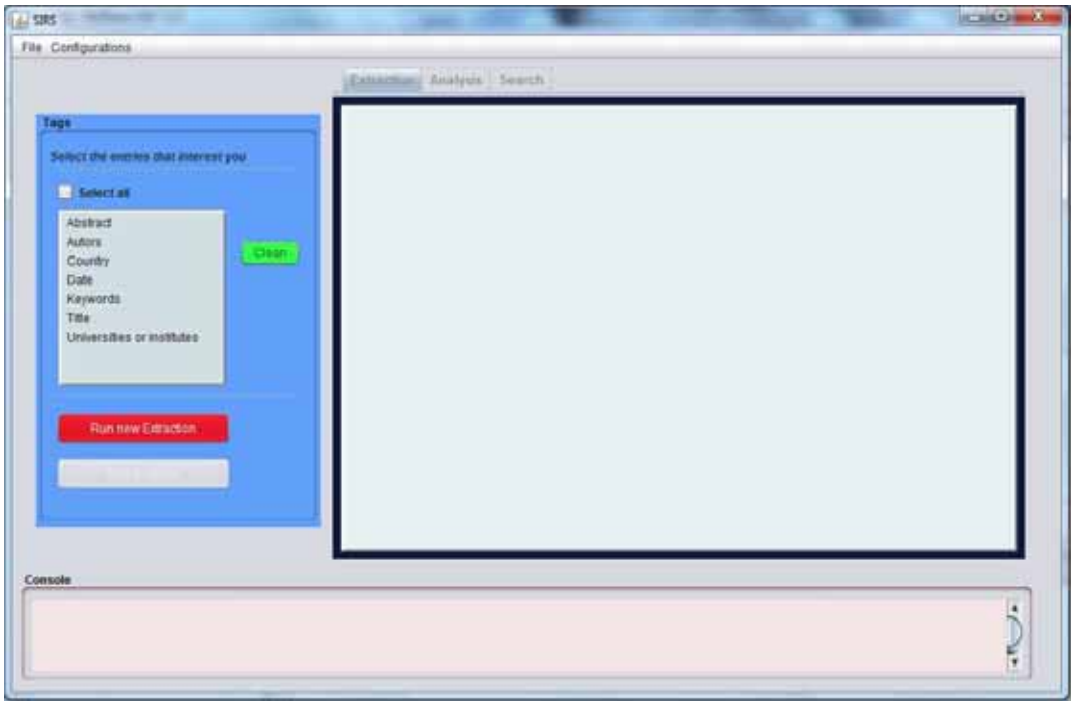
- Nº de artículos en los que recupera todos los campos correctamente: 1. (2%)
- Nº de artículos en los que no recupera un campo: 8. (21%)
- Nº de artículos en los que no recupera dos campos: 10. (26%)
- Nº de artículos en los que no recupera tres campos o más: 19. (50%)

Como medida de fiabilidad general, se ha establecido unos criterios de importancia por campo, en la que el nombre del autor, el de la publicación y la universidad se valoran con 25%, el *abstract* del documento con 15 %, y el país con 10%.

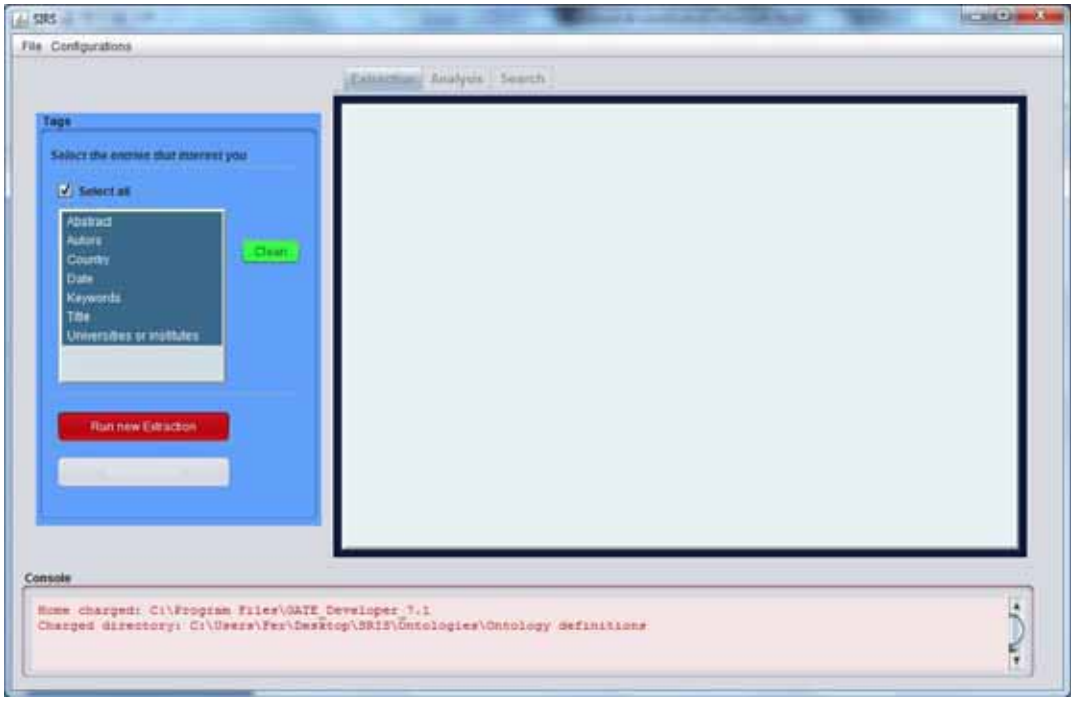
En base a lo anterior calculamos la fiabilidad general del sistema multiplicando cada recuperación válida o incompleta de cada campo por su porcentaje de importancia. Esto nos da una fiabilidad del **23.375 %.**

Manual de usuario

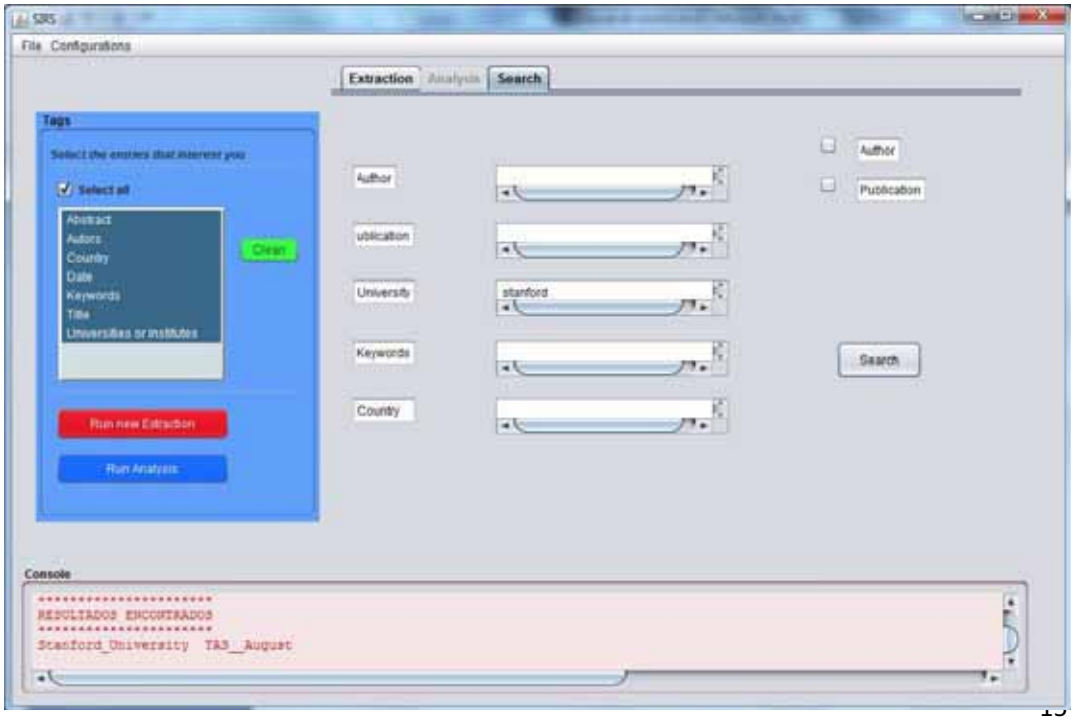
Siendo una extensión de SRIS, el funcionamiento de la aplicación desarrollada depende totalmente de él, de manera que para la ejecución del proyecto hemos de arrancar SRIS. Para que funcione el proyecto es imprescindible estar conectado a internet, porque de otro modo no cargarán las ontologías y será imposible la búsqueda.



A continuación seleccionamos las carpetas. Primero el directorio de instalación de Gate en *Configurations* y después el directorio donde se encuentran los textos para formar el *corpus*. Seleccionamos todos los campos y ejecutamos la extracción.



Para ejecutar la búsqueda, seleccionamos la pestaña *Search* y en alguno de los espacios de texto introducimos la palabra de búsqueda. La aplicación mostrará abajo en la consola los resultados obtenidos.



Conclusiones

Considero muy satisfactorio el funcionamiento y desempeño del proyecto realizado, porque recoge todas las características y objetivos específicos descritos en la propuesta del proyecto. Como comenté en el capítulo de diseño, el proyecto queda determinado no solo en la concepción e implementación, sino también en su fiabilidad y rendimiento, ya que según se ha visto en el capítulo de pruebas, la herramienta SRIS de la que partía el trabajo desarrollado en este proyecto tenía un grado de fiabilidad no demasiado alto.

Sin embargo, este módulo de inserción en ontologías, bases de datos y su posterior recuperación mediante el buscador, puede ser adaptado y usado en otro sistema de recuperación de datos diferente a SRIS. No obstante, el parseo y reconocimiento semántico de textos de lenguaje natural es tremendamente complicado.

En años anteriores en la universidad había trabajado algo con ontologías, pero no muy profusamente, de manera que este trabajo con Protegé, diseño de la ontología, revisión de código modificado y ejecución de reglas de inferencia me ha sido muy útil para comprender la gran potencia de almacenamiento y razonamiento de la que gozan las ontologías.

También ha sido interesante descubrir la gran cantidad de APIs desarrolladas enfocadas en las ontologías lo que sirve para darse cuenta que es un recurso en auge en una sociedad cada vez más y más concienciada en obtener con gran celeridad información muy variada.

Las ontologías proporcionan una capacidad casi total de diseño al usuario, y eso las hace muy útiles para moldear cualquier tipo de universo.

Encontré no pocos problemas en el desarrollo de mi trabajo. El más importante y más tiempo me costó resolverlo fue el de la escritura en la ontología. Una vez escrito una serie de artículos durante una ejecución, GATE no era capaz de parsear la ontología ya escrita, de modo que era imposible añadir más información. Se solucionó gracias a crear un template vacío desde Protegé, y desde ahí ya se pudo insertar todos los artículos que se quisiese.

Un problema de capacidad de cómputo fue el análisis de una cantidad algo mayor de artículos en una sola ejecución, ya que mi máquina carecía de memoria suficiente y no podía continuar la ejecución de SRIS. Por ello es recomendable para máquinas no muy potentes acotar el número de artículos a leer por ejecución a 10. De cualquier modo la ejecución del módulo trabajado no ocupa ninguna memoria y no genera ningún tipo de problema.

Como conclusión final considero que ha sido una buena experiencia haber podido trabajar con proyectos anteriores, ya que esto fomenta comprender otros códigos y a darse uno cuenta de que es posible un trabajo en cadena que se vaya mejorando con cada proyecto y que en un futuro pueda servir para avanzar en este campo.

Bibliografía

[1] Protégé. Available: http://protege.stanford.edu/

[2] Jess. Available: http://herzberg.ca.sandia.gov/

[3] Ugalde, Selene. "Sistema de recuperación de información semántico", proyecto terminal, División de CBI, Universidad Autónoma Metropolitana Azcapotzalco, México, 2012.

[4] GATE. http://gate.ac.uk/

[5] UML. http://www.uml.org/

[6] DIA. http://projects.gnome.org/dia/

[7] SWRL. http://www.w3.org/Submission/SWRL/, http://protege.cim3.net/cgi-bin/wiki.pl?SWRLTab

[8] OWL. http://www.w3.org/2007/OWL/wiki/OWL_Working_Group

[9] MySQL. http://www.mysql.com/

[10] NetBeans. https://netbeans.org/

[11] OWLAPI. http://owlapi.sourceforge.net/