Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Un prototipo para Question Answering bilingüe closed y open domain

Tesis presentada para optar al título de
Licenciado en Ciencias de la Computación

Julián Peller

Director: José Castaño

Buenos Aires, 2013

UN PROTOTIPO DE QUESTION ANSWERING BILINGÜE PARA CLOSED Y OPEN DOMAIN

El objetivo del proyecto es el diseño y la implementación de un sistema de question answering closed y open domain bilingüe con facilidad para la integración de nuevas fuentes de información y nuevos idiomas.

Palabras claves: Question Answering, Closed Domain, Open Domain, Multilenguaje, Information Retrieval, Question processing, Procesamiento del lenguaje natural

Índice general

1. INTRODUCCIÓN

El objetivo del proyecto es el diseño y la implementación de un sistema de question answering closed y open domain bilingüe con facilidad para la integración de nuevas fuentes de información y nuevos idiomas.

1.1. Marco general

1.1.1. ¿Qué es Question Answering?

Question Answering es un área de ciencias de la computación que combina herramientas de búsqueda y recuperación de la información (*information retrieval*) y de procesamiento del lenguaje natural (*nlp*), buscando generar respuestas a preguntas formuladas en algún lenguaje humano. Por ejemplo, para el input *"¿Cuándo nació Noam Chomsky?"* un sistema de QA debería devolver "*7 de diciembre de 1928*". El dominio de problemas está ampliamente vinculado con la *web semántica*, ya que en ambos lo que se busca es dotar de información semántica los datos "planos" disponibles en un corpus y también escribir sistemas capaces de manejar nociones semánticas al tratar con estos datos.

1.1.2. Information Retrieval

El problema conocido como ***information retrieval*** consiste, formalmente, en retornar *información relevante* para una *consulta* (*information need*) a partir de una *base de conocimiento*. El caso de uso típico, ilustrado por los motores de búsqueda web, consiste en devolver una lista de documentos relevantes, en orden de relevancia, para una consulta del usuario. En estos sistemas, las consultas se interpretan como una serie de tokens concatenados con distintos operadores lógicos binarios, donde la barra espaciadora funciona como or inclusivo. El core del sistema es un *índice de búsqueda* de los documentos de la base de conocimiento, que podemos pensar como un diccionario indexado de palabras en documentos (más adelante tocaremos este tema). El proceso de creación y mantenimiento de estos índices depende del tipo y del dinamismo de la base de conocimiento, y oscila entre un modesto setup a mano de una lista de documentos estáticos hasta el ejército de *spiders* de google indexando la web en tiempo real.

1.1.3. Natural Language Processing

El problema de question answering puede pensarse como un problema de information retrieval, pero su objetivo no es devolver una lista de documentos para una serie de palabras, sino una respuesta puntual a una pregunta puntual. Para esto incorpora una dimensión semántica, esto es, toda una serie de problemas de procesamiento de lenguaje natural que se abordan con varias herramientas, que aportan distintos modelos de análisis lingüístico a nivel *oración*, tanto de la consulta del usuario, como de los documentos indexados. Por ejemplo: el pos-tagger es un analizador que genera la estructura gramatical

de la oración, mientras que el reconocimiento de entidades nombradas (NER) identifica entidades como "Buenos Aires", "José Pérez", etc- y las clasifica (como *lugar* y *persona*, respectivamente) . Los sistemas de QA utilizan distintas herramientas para modelar la consulta y para buscarla, con algún grado de comprensión semántica, dentro de su base de conocimiento .

1.1.4. Distintos problemas

Los sistemas de QA suelen ser sistemas complejos que abordan distintas problemáticas: por un lado deben definir y optimizar la base de conocimiento para el dominio dado, por otro deben realizar un análisis de las preguntas en lenguaje natural a fin de volverlas tratables computacionalmente y, finalmente, deben poder buscar y decidir la mejor respuesta para la pregunta ingresada, si es que esa respuesta existe.

Algunos de estos subproblemas tienen nombre propio en la literatura. Por ejemplo, dependiendo de la amplitud de la base de conocimiento, un sistema puede ser *closed domain*, si la base es acotada, y *open domain*, si no lo es (en la práctica, infinito=Internet). Por su parte, bases de conocimiento más pequeñas en general requieren y permiten un modelado más exhaustivo de los datos y un análisis más estructurado, mientras que bases de conocimiento más abiertas suelen tener un enfoque apoyado más fuertemente en la estadística.

Otra distinción usual contempla el tipo de datos de la base de conocimiento: este puede ser estructurado, como en una base de datos relacional, semi estructurado, como los documentos xml, o también sin estructura, como el texto plano. Cada tipo de datos tiene su enfoque: los datos estructurados definen una ontología acotada que limita qué cosas se pueden preguntar y qué cosas se pueden responder: el problema en este caso consiste traducir la pregunta a una query definida en por el modelo de datos. Por otro lado, si los datos no tienen estructura, no es posible definir una ontología rígida y se hace necesario otro tipo de enfoque más difuso y basado en análisis lingüísticos del corpus de datos mismo contra la pregunta. No siempre, pero en general una base de conocimiento closed domain está asociada a un tipo de datos estructurado o semi-estructurado, mientras que las bases open domain suelen ser no estructuradas.

1.2. Estructura del proyecto

- Qué hace este proyecto,
- cual es el output,
- qué pretendió hacer
- qué son las dos subsecciones que siguen

1.2.1. Mitic

Este proyecto está vinculado con un marco de trabajo común entre el grupo GALLI, del DC y la fundación Sadovsky, relacionado con la construcción de una base de conocimiento y la recuperación de la información en el dominio de la investigación y la producción relacionada con TICs en argentina. La base de conocimiento generada es estructurada y *closed domain*, y consta de una serie de entidades: investigadores, universidades, publicaciones, proyectos, empresas y temáticas, con diferentes tipos de relaciones. Esta base de conocimiento está definida como un grafo de relaciones con diferentes pesos y valores entre distintas entidades, a partir de lo que originalmente era una recopilación heterogénea datos en xml. El grafo de mitic tiene relaciones directas y también inferidas, con un cierto grado de confiabilidad (peso) para cada relación. Tomando como punto de partida esta base de conocimiento, este proyecto se propone construir un sistema de question answering bilingüe de dominio cerrado. Para esto, definimos sobre la base de conocimiento una ontología rígida, que determina el conjunto de preguntas tratables, y utilizamos herramientas de procesamiento de lenguaje natural para clasificar las consultas del usuario dentro de este esquema. Para la ontología , definimos una interfaz única para las entidades que permite la integración de nuevos tipos con un esfuerzo mínimo. Esta ontología es formal y no tiene idioma. Para la traducción o el mappeo de la pregunta a esta esquema utilizamos distintos analizadores de distintas librerías.

- Idiomas
- XML a mongo como "versión más nueva y potente"

1.2.2. Clef '07

Por su parte, para desarrollar y evaluar mecanismos de dominio abierto resolvimos algunos ejercicios de la competencia de QA organizada por CLEF en 2007. CLEF (de *Cross-Language Evaluation Forum*) es una organización que busca fomentar la investigación en sistemas de information retrieval cross-language. En particular, una vez por año CLEF lanza una competencia de Question Answering multilenguaje, con diferentes corpus y diferentes tipos de ejercicios. Estas competencia permiten obtener un patrón estándar de comparación entre distintos desarrollos y una idea general del estado de arte alcanzado en cada área. Por ejemplo, la competencia finalizada del año 2013, QA4MRE@CLEF2013 [1], (Question Answering for Machine Reading Evaluation) tiene estos ejercicios y evalua Machine Reading que es esto y el link está en esta referencia (http://celct.fbk.eu/QA4MRE/).

- Diferentes competencias. Por qué la 07.
- Todos los ejercicios de 2007. Diferentes idiomas.
  - The subtasks were both:
  - monolingual, where the language of the question (Source language) and the language of the news collection (Target language) were the same;
  - cross-lingual, where the questions were formulated in a language different from that of the news collection.
- Que ejercicios uso y dar ejemplos.

- ¿Quién es Harry Potter?

Ten source languages were considered, namely, Bulgarian, Dutch , English, French, German, Indonesian, Italian, Portuguese, Romanian and Spanish. All these languages were also considered as target languages, except for Indonesian, which had no news collections available for the queries.

1.2.3. Investigación y herramientas utilizadas

El sistema está escrito en java y utiliza distintas tecnologías: mongodb y lucene para modelar la base de conocimiento, las herramientas de nlp de freeling y de stanford para analizar la pregunta... [[EXPANDIR]] [[Enumerar, brevemente, las tecnologias utilizadas y tambien los sistemas de QA que usamos como guia]]

2. MARCO TEÓRICO

2.1. Terminología

2.1.1. Tokens

Un token es una palabra o un signo de puntuación en el contexto de un texto.

Por ejemplo, el texto "El sol brilla." tiene cuatro tokens: {'El', 'sol', 'brilla', '.' }. Las herramientas que generan una lista de tokens a partir de un texto se llaman *tokenizers* y suelen permitir distintos tratamiento de ciertas palabras o signos de puntuación. Las bibliotecas que proveen tokenizers proveen también herramientas para dividir un texto en oraciones. Estos dos procesos son el primer paso de todos los análisis de procesamiento de lenguaje natural siguientes.

2.1.2. N-Gramas

Un n-grama es una subsecuencia continua de n tokens de un string.

Por ejemplo, la oración: "Hoy está nublado", tiene los siguientes n-gramas:

Tres unigramas	:	{ "Hoy", "está", "nublado" }
Dos bigramas	:	{ "Hoy está", "está nublado" }
Un sólo trigrama	:	{ "Hoy está nublado" }

Los n-gramas son útiles para recorrer un texto con distintos tamaños de ventana en busca de algún patrón conocido, por ejemplo: buscando entidades nombradas que no fueron reconocidas por otras herramientas.

2.1.3. Ontología

El término 'ontología' es un término originalmente filosófico, que refiere al estudio de lo que hay, de lo que es, o, dicho de otro modo, a la definición y al estudio de los entes que existen en la realidad última y de sus cualidades esenciales y sus relaciones intrinsecas. Aplicado a ciencias informáticas, principalmente dentro áreas como inteligencia artificial y representación del conocimiento, esta noción original se sostiene, acotando la noción de realidad última a uno o varios dominios de problemas. Más concretamente, la noción de ontología en informática refiere a la definición formal y exhaustiva de un conjunto de conceptos que representan entidades, tipos o clases de entidades, propiedades y relaciones entre estas entidades relevantes para el modelado de un dominio de problemas dado. Esta ontología formal define un vocabulario inicial fijo que determina el tipo de problemas que se pueden plantear (y resolver) para el dominio.

2.2. Herramientas

2.2.1. Indice invertido

Un índice invertido es la estructura de datos típica utilizada en problemas de information retrieval. Consiste en un mapeo de términos en documentos que los contienen. Es decir, para un término dado, un índice invertido devuelve una lista de los documentos cargados que lo contienen. Este tipo de estructuras invierte la relación normal en la cual a partir de un documento se accede a la lista de términos que este documento contiene (de allí el nombre *invertido*). Por ejemplo, para los textos:

T[0] = "qué es esto"
T[1] = "esto es un ejemplo"
T[2] = "qué gran ejemplo"

Un índice invertido contendría las siguientes entradas (dónde el número n es un puntero al texto T[n]):

"qué" : {0, 2}
"es" : {0, 1}
"esto" : {0, 1}
"un" : {1}
"ejemplo" : {1, 2}
"gran" : {2}

2.2.2. Part-of-speech (POS) tagging

El POS-tagging o *etiquetado gramatical* consiste en asignar a los diferentes tokens una etiqueta con el rol o categoría gramatical que cumplen en su contexto de uso (por lo general, una oración o un párrafo).

Por ejemplo, para la oración:

"El hombre bajó la escalera."

El resultado de un POS-tagger podría ser el siguiente:

Token	Etiqueta Gramatical (POS-tag)
El	Determinante, artículo definido, masculino, singular
hombre	Nombre común masculino singular (sustantivo)
bajó	Verbo principal indicativo pasado tercera persona del singular (genero indefinido)
la	Determinante, artículo femenino singular
escalera	Nombre común femenino singular (sustantivo)
.	Punto final

En el scope de este proyecto, utilizamos dos tipos de pos-tags: los verbos y las qwords. Las qwords son los pronombres interrogativos: qué, quién, cómo, etc. y en inglés, who, when, where...

[[Expandir]]

2.2.3. Named Entity Recognition and Classification (NER, NEC, NERC)

El reconocimiento de entidades nombradas (NER, de Named Entity Recognition) es una subtarea de Information Extraction. Information Extraction es, brevemente, todo el dominio de problemas vinculado con la extracción de información estructurada a partir de datos no estructurados o semi estructurados. NER es, dentro de este dominio, el proceso de reconocer unidades de información (las entidades nombradas) tales como nombres de personas, organizaciones, lugares, expresiones numéricas como tiempo, fechas, dinero, porcentajes, etc. A veces se habla de NERC (Named Entity Recognition and Classification) para poner énfasis en la asignación de un tipo (por ejemplo: nombre de empresa) a la entidad nombrada reconocida.

Los primeros sistemas de NER eran algoritmos basados en reglas hardcodeadas, mientras que los más modernos incorporan técnicas de machine learning y son, en general, algoritmos basados en features (ciertos aspectos de los tokens).

El primer sistema data de 1991 y constaba de reglas escritas a mano y heurísticas simples. Recién en 1996, con el estímulo de la MUC-6 (una conferencia reconocida en el área que dedicó una edición a NER), el área comenzó a acelerar su crecimiento.

[[En construcción en documento NER de GDrive]]

2.2.4. Relation Extraction

La detección y extracción de relaciones es una tarea de procesamiento de lenguaje natural que consiste en extraer entidades y relaciones semánticas entre entidades a partir de un texto no estructurado.

[[Escribir sobre distintos algoritmos conocidos basado en el paper RE1]]

2.2.5. Question Classification

Un clasificador es una herramienta que asigna a un elemento una de k clases. La clasificación, en general, es un área bastante fecunda de NLP y, más en general, de machine learning. Los clasificadores de preguntas son herramientas que clasifican preguntas, en general, según su tipo de respuesta esperada. Por ejemplo: "¿Quién descubrió América?" espera, más allá del nombre concreto, *un nombre de persona*; "¿Cuándo se descubrió América?" espera *una fecha* (o, más en general, *un tiempo*), "¿Dónde se descubrió América?" espera, como respuesta, *un lugar*, etc. Este es un eje de clasificación conocido como tipo de respuesta esperado, aunque existen otros.

Por otro lado, el tipo de clasificador puede ser estricto, i.e., asignar sólo una clase a cada objeto o bien probabilístico, i.e.: asignar un cierto grado de pertenencia a cada una de las clases posibles a cada objeto. [[Expandir]]

2.2.6. Detección de idiomas

La detección o identificación de idioma es la tarea de reconocer el idioma de un cierto contenido. Esta tarea puede pensarse como una tarea de clasificación donde las clases son los distintos idiomas que el detector puede reconocer.

[[Expandir con detalles técnicos de wikipedia]]

2.2.7. Traducción

3. ESTADO DE ARTE

En esta sección vamos a pasar revista de una serie de tecnologías investigadas o testeadas durante la tesis. La primera, IBM-Watson es quizás la más conocida para el público en general.

//write stuff

3.1. IBM-Watson

Watson es un sistema diseñado por IBM con el objetivo de competir en tiempo real en el programa de televisión estadounidense Jeopardy, logrando resultados del nivel de los campeones humanos de este programa.

El proyecto demoró 3 años de investigación, en los cuales se logró obtener la performance esperada (nivel humano experto) en cuanto a precisión, confiabilidad y velocidad, logrando derrotar a dos de los hombre con mayores récords históricos del show en un programa en vivo [MÁS DATOS -LINKS]

El objetivo del proyecto puede considerarse una extensión de lo que fue Deep Blue, el sistema que logró el nivel de los expertos humanos en el ajedrez, porque buscó superar un reto que significativo y visible del campo de la Inteligencia Artificial tanto para la comunidad científica como para la sociedad en general: "¿puede un sistema computacional ser diseñado para competir con los mejores hombre en alguna tarea que requiera altos niveles de inteligencia humana y, si es el caso, que clase de tecnología, algoritmos e ingenieria se requiere?"[1]

Watson es la implementación específica para participar en este programa de una arquitectura más genérica de question answering, DeepQA, que da el nombre al proyecto de la corporación. Esta arquitectura ejemplifica perfectamente la complejidad del problema de QA de dominio abierto e incorpora tecnologías de punta de distintos dominios de ciencias de la computación, y de IA en particular: information retrieval, natural language processing, knowledge representation and reasoning, machine learning e interfaces humano - computadora.

3.1.1. El problema

Watson debe realizar tareas como parsing, question classification, question decomposition, automatic source adquisition and evaluation, entity and relation detection, logical form generation, knowledge representation and reasoning manteniendo ciertos atributos de calidad bastante exigentes derivados de la naturaleza del show. Estas restricciones son:

- Confiabilidad de la respuesta:
  Jeopardy tiene tres participantes con un pulsador y el que desee responder debe pulsar antes que los demás. Además, existe una penalización por respuestas incorrectas, por lo que es esencial que el sistema pueda determinar la confiabilidad de la respuesta obtenida a fin de optar por responder o no responder.

[1] Traducción propia de Building Watson: An Overview of the DeepQA Project, p2

- Tiempos de respuesta:
  La confiabilidad de la respuesta, o al menos una estimación, debe calcularse antes de que pase el tiempo para decidir responder (6 segundos) y también de que otro participante oprima su pulsador (menos de 3 segundos).

- Precisión:
  El tipo de respuestas que se dan en el show suelen ser respuestas exactas (por ejemplo: solamente un nombre, un número o una fecha, etc).

El sistema cuenta con varios componentes heurísticos que estiman ciertos features y grados de confiabilidad para diferentes respuestas, los cuales son evaluados por un sistema general que sintetiza un grado de confiabilidad para una respuesta final y determina así si responder o no responder.

El programa consta de un tablero con 30 pistas (o preguntas) organizadas en seis columnas, cada una de las cuales es una categoría. Las categorías van desde temas acotados como "historia" o "ciencias" hasta temas más amplios como "cualquier cosa" o "potpourri". Watson intenta respuestas sobre varias hipótesis de dominio y verifica en cual de ellos se logran respuestas de mayor confiabilidad.

Por otra parte, el grueso de las preguntas de Jeopardy son del tipo *factoid*, esto es, preguntas cuya respuesta esta basada en información fáctica acerca de una o más entidades individuales.

Por ejemplo:
Categoría: Ciencia General
Pista: Cuando es impactado por electrones, un fósforo emite energía electromagnética de esta forma
Respuesta: Luz (o fotones)

A su vez, existen ciertos tipos de pistas que requieren un enfoque particular, por ejemplo, pistos que constan de dos subpistas muy débilmente relacionadas, o problemas matemáticos formulados en lenguaje humano, o problemas de fonética, etc, que no pueden ser simplemente dejados de lado porque, si bien tiene poca probabilidad de aparición, cuando aparecen lo hacen en bloque y pueden arruinar el juego de Watson. Se acordó con la productora del programa, sin embargo, dejar de lado preguntas audiovisuales (aquellas que presentan una imagen o un audio y requieren interpretarlo) y preguntas que requieren instrucciones verbales del presentador.

Para determinar el dominio de conocimiento, los investigadores analizaron 20000 preguntas, extrayendo su LAT (lexical answer type, o tipo léxico de respuesta). El LAT se define como una palabra en la pista que indica el tipo de la respuesta esperado. Por ejemplo, para la pista "Investanda en 1500's para agilizar el juego, este movimiento involucra dos piezas" el LAT es "movimiento". Menos del 12 % de las pistas no indicaba explícitamente ningún LAT, usando palabras como "esto" o "eso". En estos casos, el sistema debe inferir el tipo de respuesta del contexto. Del análisis de estas 20000 pistas se reconocieron 2500 tipos léxicos distintos, de los cuales los 200 más frecuentes no llegaban a cubrir el 50 % del total de pistas. Esto implica que un approach estructurado (orientado por el tipo de respuesta), si bien resulta útil para algunos tipos, no es suficiente para abordar el problema completo.

3.1.2. Métricas

Las métricas de resultados, además del tiempo de respuesta, son la *precisión* (preguntas contestadas correctamente / preguntas contestadas) y el *porcentaje de respuestas dadas* (preguntas contestadas / total de preguntas). Mediante la configuración de un threshold de *confiabilidad* pueden obtenerse distintas estrategias de juego: un umbral bajo repercutirá en un juego más agresivo, incrementando la proporción de respuestas contestadas,, pero disminuyendo su precisión, mientras que un umbral alto determinará un juego conservador, con menos respuestas dadas pero mayor precisión en las mismas. Es un clásico escenario de trade-off entre dos atributos de calidad. Un buen sistema de estimación de confiabilidad implica una mejora general del sistema, aún cuando el módulo de generación de respuestas permanezca idéntico.

En el show, el porcentaje de respuestas dadas depende de la velocidad con la que se llega a presionar el pulsador, lo cual sólo interesa para el dominio de QA como una restricción temporal.

Mediante análisis numérico, los investigadores determinaron que los campeones de Jeopardy lograban tomar entre el 40 % y el 50 % de las preguntas y, sobre ellas, lograban una precisión de entre el 85 % y el 95 %, lo que determinaba una barrera de performance bastante exigente en lo que respecta a QA.

3.1.3. Baseline

El equipo de IBM intentó utilizar dos sistemas consolidados en QA y adaptarlos al problema de Jeopardy. El primero fue PIQUANT (Practical Intelligent Question Answering Technology), un sistema desarrollado por IBM en conjunto con el programa del gobierno estadounidense AQUAINT y varias universidades, que estaba entre los mejores según la TREC (Text Retrieval Conference), una autoridad en el área. PIQUANT consta de un pipeline típico (véase QANUS) con tecnología de punta, logrando un rango del 33 % de respuestas correctas en las evaluaciones TREC-QA. Los requerimientos de la evaluación de TREC son muy distintos de los de Jeopardy: TREC ofrece un corpus de conocimiento relativamente pequeño (1M de documentos) de donde las respuestas deben ser extraídas y justificadas, el tipo de preguntas de TREC son menos complejas a nivel lingüístico que las de Jeopardy y la estimación de confiabilidad no resulta una métrica importante (dado que no hay penalización por respuestas incorrectas). Además, los sistemas tienen permitido acceder a la web y las restricciones temporales son, por mucho, más amplias (por ejemplo: una semana para responder 500 preguntas). En Jeopardy, además de las restricciones ya mencionadas, un requerimiento fue que el sistema trabaje sobre datos locales y no acceda a la web en tiempo real. El intento de adaptar PIQUANT al problema de Jeopardy dio pésimos en comparación con los necesarios: 47 % de precisión sobre el 5 % de respuestas con mayor confiabilidad y 13 % de precisión en general.

Por otro lado, el equipo intentó adaptar el sistema OpenEphyra (véase OpenEphyra), un framework open-source de QA desarrollado en CMU (Carnegie Mellon University) basado en Ephyra (no libre), diseñado también para la evaluación TREC. OpenEphyra logra un 45 % de respuestas correctas sobre el set de datos de evaluación TREC 2002, usando busqueda web. La adaptación resultó aún peor que la de PIQUANT (con menos del 15 % de respuestas correctas y una mala estimación de la confiabilidad).

Se probaron dos adaptaciones de estos sistemas. una basada en búsquedas de texto puro y otra basada en reconocimiento de entidades. En la primera, la base de conocimiento se modeló de manera no estructurada y las preguntas se interpretaron como términos de una query, mientras que en la segunda se modeló una base de conocimientos estructurada y las preguntas se analizaron semánticamente para reconocer entidades y relaciones, para luego buscarlos en la base. Comparando ambos enfoques en base al porcentaje de respuestas dadas, el primero dio mejores resultados para el 100 % de las respuestas, mientras que la confiabilidad general era baja; por otro lado, el segundo enfoque logró altos valores de confiabilidad, pero sólo en los casos en que efectivamente logra identificar entidades. De aquí se infiere que cada enfoque tiene sus ventajas, en el dominio de problemas apropiado.

3.1.4. La arquitectura DeepQA

Los intentos de adaptación iniciales, como vimos, no dieron resultados, así como tampoco sirvieron las adaptaciones de algoritmos de la literatura científica, los cuales son realmente difíciles de sacar de su contexto original y de las evaluaciones sobre las cuales fueron testeados. Este problema, veremos -por ejemplo, con QANUS y Reverb- , se repitió en nuestro proyecto. Como conclusión de estos intentos frustrados, el equipo de IBM entendió que una arquitectura de QA no debía basarse en sus componentes concretos sino en la facilidad para incorporar nuevos componentes y para adaptarse a nuevos contextos. Así surgió DeepQA, la arquitectura de base, de la cual Watson es una instancia concreta para un contexto particular (con requerimientos de alta precisión, buena estimación de confiabilidad, lenguaje complejo, amplitud de dominio y restricciones de velocidad). DeepQA es una arquitectura de computo paralelo, probabilistico, basado en recopilación de evidencia y scoring. Para Jeopardy se utilizaron más de 100 técnicas diferentes para analizar lenguaje natural, identificar y adjudicar valor a fuentes de información, encontrar y generar hipótesis, encontrar y rankear evidencias y mergear y rankear hipótesis en función de esta evidencia. La arquitectura sirvió para ganar Jeopardy, pero también se adaptó a otros contextos como la evaluación TREC, dando resultados mucho mejores que sus predecesores. Los principios de diseño subyacentes de la arquitectura son:

- Paralelismo masivo
  Para evaluar distintas hipótesis en distintos dominios con poco acoplamiento.
- Pervasive confidence estimation:
  Ningún componente genera la respuesta final, sino que da una serie de features y grados de confiabilidad y evidencia para distintas hipótesis, que luego son sintetizados.
- Integrate shallow and deep knowledge:

A continuación, enumeraremos la lista de pasos que sigue el sistema para obtener la respuesta a una pregunta:

3.1.4.1. Adquisición de contenidos

El primer paso de DeepQA es la adquisición de contenidos. Este paso es el único que no se realiza en tiempo de ejecución y consiste en crear la base de conocimiento en la cual

el proceso final buscará la respuesta a la pregunta, combinando subprocesos manuales y automáticos.

En principio se caracteriza el tipo de preguntas a responder y el dominio de aplicación. El análisis de tipos de preguntas es una tarea manual, mientras que la determinación del dominio puede encararse computacionalmente, por ejemplo, con la detección de LATs que señalamos antes. Dado el amplio dominio de conocimientos que requiere Jeopardy, Watson cuenta con una gran cantidad de enciclopedias, diccionarios, tesauros, artículos académicos y de literatura, etc. A partir de este corpus inicial, el sistema busca en la web documentos relevantes y los relaciona con los documentos ya presentes en el corpus.

Además de este corpus de documentos no estructurados, DeepQA maneja contenidos semi-estructurados y estructurados, incorporando bases de datos, taxonomías y ontologías como dbPedia, Wordnet y las ontologías de Yago.

3.1.4.2. Análisis de la pregunta

El primer paso en run-time es el análisis de la pregunta. En este paso el sistema intenta entender qué es lo que la pregunta está preguntado y realizar los primeros análisis que determinan cómo encarará el procesamiento el resto del sistema. Watson utiliza shallow parses, deep parses, formas lógicas, pos-tags, correferencias, detección de entidades nombradas y de relaciones, question classification, además de ciertos análisis concretos del domiento del problema.

En este proceso se clasifica el tipo de la pregunta (los tipos están determinados por el show: puzzles, matemáticos, etc). También se busca el tipo de respuesta esperada, dónde los tipos manejados son por Watson son los LATs extraídos de las preguntas de ejemplo. El LAT determina el "tipo" de la respuesta, que clase de entidad *es* la respuesta (una fecha, un hombre, una relación, etc). El equipo de IBM intentó adaptar distintos algoritmos de clasificación preexistentes, pero después de intentar entrenarlos para el dominio de tipos de Jeopardy, llegaron a la conclusión de que su eficacia era dependiente del su sistema de tipos default, y que la mejor forma de adaptación era mappear su output a los tipos utilizados por Watson (un enfoque similar fue utilizado en esta tesis con respecto al clasificador de Stanford). Otra anotación importante es el "foco" de la pregunta, la parte de la pregunta tal que si se la reemplaza por la respuesta, la pregunta se convierte en una afirmación cerrada.

Por ejemplo, para "El hombre que escribió Romeo y Julieta", el foco es "El hombre que". Este fragmento suele contener información importante sobre la respuesta y al reemplazarlo por una respuesta candidata se obtiene una afirmación fáctica que puede servir para evaluar distintos candidatos y recolectar evidencia. Por ejemplo, reemplazando por distintos autores y verificando que la oración resultante esté presente en el corpus.

Por otro lado, muchas preguntas involucran relaciones entre entidades y, más puntualmente, tienen una forma sujeto-verbo-objeto. Por ejemplo, tomando la pista anterior, podemos extraer la relación *escribir(x, Romeo y Julieta)*. La amplitud del dominio de Jeopardy hace que la cantidad de entidad y de relaciones entre entidades sea enorme, pero esto empeora aún más al considerar las distintas formas de expresar la misma relación. Por eso, Watson sólo logra encontrar directamente una respuesta mediante reconocimiento de entidades y relaciones sobre el 2 % de las pistas. En general, este tipo de enfoque es útil sobre dominios más acotados, mientras que la detección de relaciones como approach general a un problema de question answering de dominio amplio es un área de investigación abierta.

Una particularidad ya señalada de las preguntas de Jeopardy son las pistas con subpistas no relacionadas. Para atacar este problema, Watson genera distintas particiones y resuelve todas en paralelo, sintetizando las respuesta de cada partición generada mediante algoritmos ad-hoc de ponderación de confiabilidad y otras características.

3.1.4.3. Generación de Hipótesis

El tercer paso (segundo en run-time) es la generación de hipótesis: tomando como input el resultado del paso anterior se generan respuestas candidatas a partir de la base de conocimiento offline. Cada respuesta candidata reemplazada por el foco de la pregunta es considerada una hipótesis, que el sistema luego verificará buscando evidencias y adjudicando un cierto grado de confiabilidad.

En la búsqueda primaria de respuestas candidatas, se busca generar tantos pasajes como sea posible. El resultado final obtenido revela que el 85 % de las veces, la respuesta final se encuentra entre los primeros 250 pasajes devueltos por la búsqueda primaria. La implementación utiliza una serie variada de técnicas, que incluyen diferentes motores de búsqueda de textos (como Indri y Lucene), búsqueda de documentos y de pasajes, búsquedas en bases de conocimiento estructuradas como SPARQL con triple store y la generación de mutiples queries a partir de una sola pregunta. La búsqueda estructurada de triple stores depende del reconocimiento de entidades y relaciones del paso anterior.

Para un número pequeño de LATs, se definió una suerte de conjunto de entidades fijas (por ejemplo: países, presidentes, etc). Si la respuesta final no es retornada en este paso, entonces no hay posibilidad de obtenerla en los siguiente. Por eso se prioriza el recall sobre la precisión, con el supuesto de que el resto del pipeline logrará filtrar la respuesta correcta correctamente. Watson genera varios cientos de hipótesis candidatas en este paso.

3.1.4.4. (Soft filtering)

Para optimizar recursos, se realiza un filtrado liviano de respuestas antes de pasar a la recopilación de evidencia y al scoring de hipótesis. Un filtrado liviano es, por ejemplo, comprobar similaridad de la respuesta candidata con el LAT esperado de la respuesta. Aquellas hipótesis que pasan el filtro pasan al siguiente proceso, que realiza un análisis más exhaustivo.

3.1.4.5. Recuperación de evidencias y Scoring de hipótesis

Para recuperar evidencias se utilizan varios algoritmos. Uno particularmente útil es buscar la hipótesis candidata junto con las queries generadas por la pregunta original, lo que señala el uso de la respuesta en el contexto de la pregunta. Las hipótesis con sus evidencias pasan al siguiente paso, dónde se les adjudica un score.

El proceso de scoring es donde se realiza la mayor parte del análisis más fuerte a nivel computacional. DeepQA permite la incorporación de distintos Scorers, que consideran diferentes dimensiones en las cuales la hipótesis sirve como respuesta a la pregunta original. Esto se llevó a cabo definiendo una interfaz común para los scorers. Watson incorpora más de 50 componentes que producen valores y diferentes features basados en las evidencias, para los distintos tipos de datos disponibles (no estructurados, semi estructurados y

estructurados). Los scorers toman en cuenta cuestiones como el grado de similaridad entre la estrurctura de la respuesta y de la pregunta, relaciones geoespaciales y temporales, clasificación taxonómica, roles léxicos y semánticos que se sabe que el candidato puede cumplir, correlaciones entre terminos con la pregunta, popularidad (u obscuridad) de la fuente del pasaje, aliases, etc.

POR EJEMPLO: COPIAR NIXON

Los distintos scores se combinan luego en un score único para cada dimensión.

(Merge)

Recién después de este momento, Watson realiza un merge entre hipótesis idénticas. Las hipótesis idénticas son diferentes formulaciones lingüisticas de lo mismo, por ejemplo: "X nació en 1928" o "El año de nacimiento de X es 1928". Finalmente, se procede a estimar un ranking único y una confiabilidad única para las distintas hipótesis. En este paso se utilizan técnicas de machine learning que requieren entrenamiento, y modelos basados en scores. Se utilizan técnicas jerárquicas como mixture of experts y stacked generalization y, finalmente, un metalearner fue entrenado para ensamblar los distintos resultados intermedios.

3.1.5. Tiempos y escala

DeepQA utiliza Apache UIMA, un framework que implementa UIMA (Unestructured Information Management Architecture): todos los componentes de DeepQA son IUMA-annotators, módulos que producen anotaciones y aserciones sobre un texto.

La implementación inicial de Watson corría sobre un sólo procesador y demoraba aproximadamente 2 horas en contestar una sola pregunta. La arquitectura paralela permite, sin embargo, que al correrlo sobre 2500 núcleos -de la implementación final- los tiempos de respuesta oscilen entre 3 y 5 segundos, que es lo esperado.

Finalmente, la implementación de Watson logró alcanzar el estándar de resultados de los campeones de Jeopardy y, como ya dijimos, compitió y ganó el programa en Febrero de 2011. Además, se realizaron adaptaciones para trabajar sobre los problemas de TREC, en los cuales se demostró una amplia mejoría en comparación con PIQUANT y OpenEphyra

3.1.6. Conclusiones sobre IBM-Watson

System level approach.

3.2. QANUS

QANUS (Question-Answering @ National University of Singapore) es un sistema de question answering basado en information retrieval. El proyecto se actualizó por última vez en noviembre de 2012 y contiene las herramientas más actuales de nlp (el POS-tagger, el NER-tagger y el Question Classifier de Stanford) y también de information retrieval índice de búsquedas lucene), todo de código abierto. El código cuenta con un framework (Qanus), que cumple un rol equivalente a la arquitectura DeepQA en el proyecto anterior, y un sistema montado sobre este framework QA-sys, equivalente a Watson (Ver Figura 3.1).

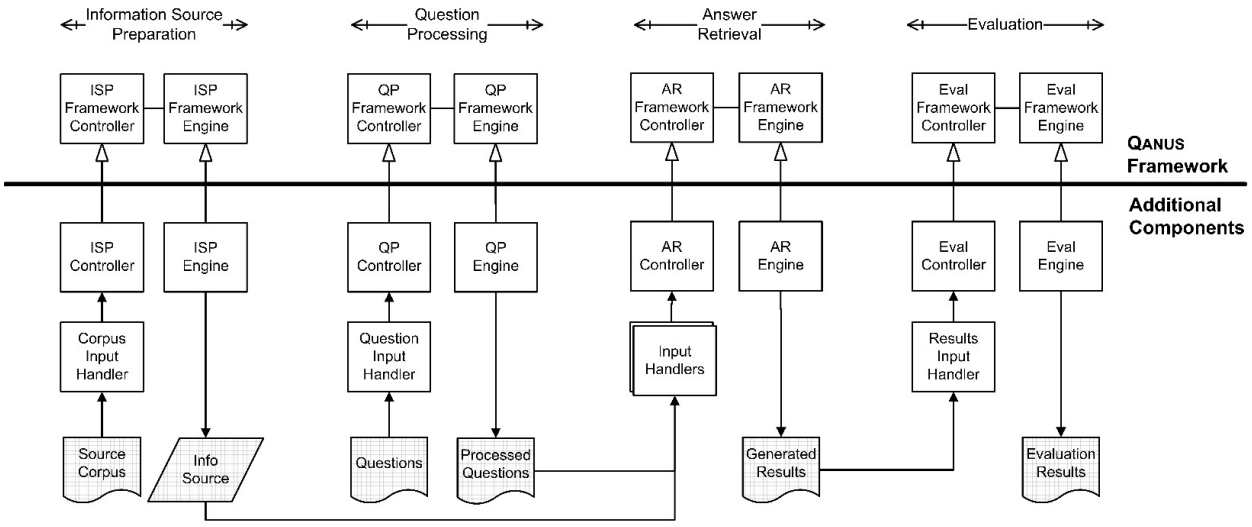

Fig. 3.1: El framework Quanus y la implementación QA-sys

La motivación de esto es proveer a la comunidad científica un framework para ingresar al mundo de QA de una manera más sencilla y rápida, permitiendo construir nuevos sistemas de QA sobre esta arquitectura. En efecto, la arquitectura DeepQA no está disponible para la comunidad, el ya mencionado OpenEphyra, como veremos en breve, no funciona, mientras que otros sistemas resultan igualmente inaccesibles (Aranea, Qanda) mientras que QA-sys es un sistema de QA out of the box. Mencionaremos los logros y los límites de estos objetivos cuando hablemos de nuestro intento por montar nuestro propio sistema sobre Qanus.

La arquitectura, al igual que la de DeepQA, es la de un pipeline. Este sistema consta de tres pasos principales, que se ejecutan todos por separado (off-line):

3.2.1. Information Source Preparation

Este paso está pensado para preprocesar cualquier base de conocimiento y dejarla preparada para el paso 3 (retorno de la pregunta). El framework se propone tan amplio que no hay más especificaciones al respecto. La implementación puntual asume una base de conocimiento en formato XML de AQUAINT[2] y la incorpora a un índice de búsquedas Lucene. Este paso incorpora todo el conocimiento que estará finalmente offline; accesos dinámicos a la web, por ejemplo, se modelan en el paso 3.

3.2.2. Question Processing

Este paso, igualmente genérico, permite la incorporación de distintos componentes para anotar los tokens de la pregunta con datos útiles para ser consumidos por el paso 3. Otro procesamiento a realizar en este paso podría ser la generación de queries entendibles

[2]

por los distintos motores de almacenamientos de información del paso 1. En particular, la implementación trae un pos-tagger, un ner-tagger y un question classiffier, todos de Stanford. Hablaremos más de estos componentes más adelante.

3.2.3. Answer Retrieval

En este paso se utiliza la información generada en la preparación de la base de información y en el procesamiento de la pregunta para generar una respuesta. También puede incorporarse accesos a la web y validaciones de las respuestas candidatas. La implementación concreta evalúa cada pasaje de los primeros n documentos retornados por Lucene para la pregunta original con una serie de componentes ad-hoc de distancia para adjudicar diferentes grados de confiabilidad a los distintos pasajes.

Además, se provee de un cuarto paso opcional (el sistema de QA está completo con los tres pasos anteriores), para la fase de desarrollo y de evaluación de la performance del sistema:

3.2.4. Evaluation

Este paso está pensado para evaluar las respuestas generadas y presentarlas de un modo conciso en la fase de desarrollo. Básicamente, cruza las respuestas obtenidas contra unas respuestas esperadas escritas a mano y presenta el total de respuestas dadas correctamente.

3.2.5. Implementación

El código está escrito en java y mantiene una interfaz común a todos los pasos: un controller cuyas responsabilidades son cargar los componentes y un engine que utiliza los componentes para leer el input, procesarlo y grabar el resultado. La adaptabilidad del framework está dada en la posibilidad de incorporar componentes respetando la interfaz especificada para los mismos o bien, en modificar esta misma interfaz. Presuntamente, el framework es lo suficientemente abierto para permitir la implementación de sistemas basados en distintas fuentes de conocimiento (ontologías, archivos, web) y con distintos modos de funcionamiento mediante poco esfuerzo de customización.

La implementación llamada QA-sys está desarrollada para correr sobre el tipo de datos de las evaluaciones TREC 2007 (XML AQUAINT). En el primer paso, incorpora los XML en este formato a un índice Lucene, en el segundo paso utiliza anota la pregunta con POS tags, NERs y clasifica el tipo de respuesta con un clasificador entrenado y luego, en el tercer paso se busca la pregunta sobre el índice lucene y se retorna una lista con n documentos rankeados. Estos documentos se subdividen en pasajes. Luego se aplican diferente algoritmos ad-hoc dependiendo del tipo de respuesta esperada. Por ejemplo, si la respuesta es un nombre de persona, se ejecuta NER sobre los diferentes pasajes buscando nombres candidatos, si el tipo esperado es una fecha, se utilizan expresiones regulares escritas a mano, etc. Finalmente, los pasajes candidatos se evalúan utilizando heurísticas de proximidad de los candidatos a la pregunta inicial. Para esto se utilizan diferentes Scorers que rankean los pasajes según diferentes características (features) y luego

se selecciona alguna priorizando algunas características sobre otras, dependiendo también del tipo de respuesta esperada. Por último, el evaluador de resultados mide la exactitud (*accuracy*): total de respuestas correctas sobre total de preguntas. QA-sys funciona sólo sobre preguntas del tipo factoid y, a modo de comparación, el mejor sistema según la TREC 2007, el LymbaPA07 obtuvo un grado de exactitud del 0.706 y el décimo (Quanta) obtuvo 0.206, mientras que QA-sys logra el 0.119. La implementación es realmente simple y funciona sólo a modo de ilustración de lo que puede construirse sobre el framework.

3.3. Otros sistemas de QA: OpenEphyra, Aranea y Just.Ask

El paper describe las arquitecturas de todos los sistemas, si sirve meter más info

El paper [EPHYRA1] busca crear un criterio cuantitativo para comparar la eficacia de distintos pasos de Just.Ask, Open Ephyra y Aranea basándose en la arquitectura *pipeline de tres pasos* compartida por todos. Los tres sistemas, por lo demás, están basados en la web, utilizando distintas APIs de buscadores o bien analizando los resultados de la interfaz de usuario de los mismos.

El primer ítem importante a destacar de este trabajo, es que, al momento de la experimentación **Aranea no funcionaba más y estaba discontinuado**[3]**.** El autor se comunicó con el responsable del proyecto que corroboró que las APIs de los buscadores en los que se basaba Aranea cambiaron y no había interés en readaptar el código para que vuelva a funcionar. Las comparaciones que logró entre Just.Ask y Open Ephyra son interesantes y concluyentes a favor de la performance de OpenEphyra.

(Freeling + Cambio de Base + Rigidez)

[3] (Resaltado en Sección 8, muy concluyente).

4. INVESTIGACIÓN Y DESARROLLO

4.1. Frameworks

4.1.1. No funcionales

Para la implementación de nuestro sistema, originalmente, evaluamos la utilización de distintos frameworks disponibles. DeepQA, el producto de IBM, no es de código abierto, por lo que acerca de su implementación sólo sabemos lo que ventilaron en sus artículos técnicos. Just.ask, el sistema basado en web comparado contra OpenEphyra no está disponible en la web al momento de escribir este trabajo, mientras que OpenEphyra no funciona tal cual está diseñado originalmente (basado en web), sino que el autor sugiere unos pasos esotéricos para configurarlo para usar conocimiento local. Cabe destacar que esta falla en la funcionalidad está asociada a la que había encontrado [AUTOR DE PAPER EPHYRA1] en Aranea y está vinculado con una serie de medidas restrictivas tomadas por las compañías de buscadores, que fueron cerrando sus accesos gratuitos para la comunidad de investigación bloqueando sus APIs y el acceso automático a sus UI. Las alternativas para el uso de buscadores, actualmente, se reducen a la configuración de una serie de proxies sobre los que rotar el acceso a la UI y así engañar al detector de accesos automáticos -alternativa de legalidad cuestionable - o bien al pago por una quota de queries por mes. OpenEphyra sobrevivió a Aranea porque sus responsables escribieron una interfaz para Bing cuando Google cerró sus puertas, mientras que los responsables de Aranea no lo hicieron. Finalmente, Bing también bloqueo el acceso automático gratuito. Notar que el mismo tipo de discontinuación ocurrió con el API de traducciones de Google. La empresa declara, explícitamente, que no está dispuesta a acceder a ninguna quota de acceso gratuito para la investigación académica y que todos sus servicios son pagos.

4.1.2. Qanus

Finalmente, un sistema que *sí* estaba disponible y funcionando fue Qanus, que respetaba al pie de la letra su detalle técnico. Al comienzo del proyecto, contábamos con un corpus de datos en XML, lo cual coincidía, al menos en gran parte, con el input esperado de la implementación Qa-sys. A pesar de esto, la adaptación de los componentes no fue nada trivial y requirió un tiempo excesivo. En particular, existían dos opciones a la hora de construir un sistema sobre la arquitectura Qanus: dejar de lado la implementación Qa-sys e implementar todos los componentes de cero sobre la arquitectura, respetando las interfaces dadas por el framework, o adaptar el sistema funcionando para que trabaje sobre los nuevos datos y el nuevo entorno esperado. Frente a esta alternativa, se aparece claro que el framework en sí mismo no aporta demasiado, pues lo único que hace es atar la implementación final a una interfaz estructurada de tres procesos bastante sencillo. Además, existe un cierto grado de dependencia de la arquitectura hacia la implementación final, quizás no a nivel técnico, pero si en el modo en el que está definida la estructura. Por este motivo, encaramos una adaptación de Qa-sys a nuestro modelo de datos y a nuestros requerimientos, pero los resultados no fueron buenos en términos de resultados por

tiempo invertido. El tiempo de aprendizaje del framework mismo y el tiempo requerido para adaptar las distintas componentes propias a las interfaces esperadas por Qanus es demasiado alto para la solución que brinda. Como recién mencionamos, en realidad, el proceso de pipeline de tres pasos no tiene tantas aristas, y adaptarse a un framework es mucho menos ameno que escribirlo. Este puede ser uno de los motivos por los cuales, como acertadamente señalan los autores de Qanus, no existe ningún framework estandarizado dentro del ámbito de la investigación en QA. Después de la investigación inicial, podríamos concluir que está estandarizado, al menos a modo conceptual, la idea de que la resolución del problema se debe enfocar como un pipeline de al menos tres pasos que incluyen:

- el preprocesamiento de la base de conocimiento,
- el preprocesamiento de la pregunta,
- el retorno de la respuesta a partir de los resultados de los dos pasos anteriores.

Como último comentario al respecto, el modelo de Qanus resultaba poco atractivo a la hora de incorporar procesamiento en varios idiomas: el mejor approach utilizando esta arquitectura era implementar dos sistemas basados en Qanus paralelos y utilizar uno u otro de acuerdo con el resultado de una detección inicial.

Si bien el modelo de Qanus fue, por los motivos recién expuestos, dejado de lado, debemos destacar una serie de puntos en los que fue útil.

En primer lugar, uno de los objetivos de Qanus es facilitar el ingreso al área del QA de nuevos investigadores. Creemos que esto está logrado perfectamente: el código es muy sencillo y claro y lo mismo ocurre con la documentación, lo que hace de Qanus un proyecto muy útil desde una perspectiva pedagógica o educativa, más allá de que sea esta misma simpleza la que más adelante atente contra la usabilidad. El modelo de pipeline, que es el enfoque teórico usual al problema, y los distintos componentes y usos típicos de estos componentes en los distintos pasos del pipeline se realizan linealmente en la implementación de Qanus y Qa-sys. En particular, la similaridad entre la descripción del código de IBM (DeepQA y Watson) y el enfoque con el que Qanus ataca el mismo problema salta a la vista, considerando la diferencia de escalas.

En segundo lugar, en el plano de la investigación del estado de arte de los sistemas de QA disponibles creemos que el intento con Qanus redundó en un cierto escepticismo sobre la posibilidad de resolver nuestro problema utilizando herramientas disponibles de gran escala. La conclusión es análoga a la que tuvo el equipo de IBM al intentar usar OpenEphyra y PIQUANT: el tiempo de customización y adaptación de los framework a nuestro problema puntual es demasiado alto en comparación con el tiempo necesario para construir una nueva arquitectura que cumpla los mismos requisitos.

Por último, en un nivel técnico, Qanus nos resultó de utilidad para construir nuestro modelo final pues reutilizamos varios de los componentes de Qa-sys: En primer lugar, recuperamos el POS tagger, el NER tagger y el Question Classiffier (QC) de Stanford, que son las librerías principales con las que Qa-sys encara el procesamiento lingüístico de la pregunta y parte del proceso de generación de respuestas. Todas estas herramientas están disponibles en la web por otros medios, pero algunas -principalmente el QC- requieren un cierto tiempo de configuración inicial que los autores de Qanus ya habían resuelto. Es decir, reutilizamos, además de estos módulos externos, bastante de la configuración y las APIs de acceso a estos módulos escritos por los singapurenses. Estas herramientas

funcionan bien sólo para inputs en inglés. Las adaptaciones que hicimos las veremos más abajo. Por otro lado, incorporamos casi sin modificaciones algunas métricas de distancia entre pasajes que Qa-sys usa en el momento de la generación de la respuesta como Scorers. Estos son las clases: **FeatureSearchTermCoverage**, **FeatureSearchTermFrequency**, **FeatureSearchTermProximity**, **FeatureSearchTermSpan**. Explicaremos esta métricas en breve, dentro de nuestro modelo, bajo en nombre de "Comparadores". Este código está escrito por los autores de Qanus (es decir, no es una librería externa utilizada por ellos).

4.2. Arquitectura

4.2.1. Motivación

Después del intento con Qanus, decidimos implementar el sistema por fuera de cualquier framework y encaramos el diseño actual. En este diseño respetamos el modelo típico de pipeline de tres pasos que abunda en la literatura científica y, por lo demás, parece el indicado a la hora de encarar este tipo de problemas. Un momento no-técnico importante a destacar es la obtención de una base de datos en mongodb, resultado del trabajo del proyecto MITIC, la cual cambió sustancialmente el enfoque anterior, basado en XMLs. A partir de estos datos, fue posible delinear un esquema de entidades formal que determinó qué se puede responder y qué no. Como vimos, la estrategia de QA cuando el tipo de datos es estructurado es radicamente distinta que la estrategia cuando los datos son no estructurados. Qanus, por su parte, está orientado a un tipo de datos no estructurados: buscar documentos rankeados en un índice de búsqueda y rastrear en ellos pasajes mediante distintos métodos. Cuando la base de conocimientos consta de un tipo de datos estructurado (esto es, de entidades, relaciones, atributos de entidades) es posible delimitar una ontología más rígida que permita concentrarse en la interpretación de la pregunta hacia un lenguaje formal. El arquetipo de este enfoque puede pensarse como la traducción de un lenguaje de consulta humano a un lenguaje de consulta formal, como por ejemplo, SQL: esta estrategia de QA puede entenderse como una interfaz inteligente a una base de datos. En eje principal en este acercamiento está en el análisis lingüístico de la pregunta a fin de mapearla a un dominio conocido y, por otro lado, no es necesario hacer análisis lingüístico sobre el corpus de datos.

Por otro lado, dado que nuestro objetivo inicial incluía desarrollar métodos de QA con soporte bilingüe basados en textos (datos no estructurados) y esta base de conocimiento no lo permite, incorporamos al proyecto la resolución algunos de los ejercicios de la competencia QA4MRE (Question Answering for Machine Reading Evaluation) de la CLEF (Cross-Language Evaluation Forum) del año 2007.

Estos ejercicios fueron elegidos para poder evaluar nuestros métodos de análisis lingüsticos, ya que la naturaleza del dominio de conocimiento del proyecto mitic - por el formato de sus datos y por lo puntual de su temática- hace imposible cualquier métrica de evaluación objetiva. Tras investigar distintas competencias y métodos de evaluación, optamos por la CLEF del 2007 por contar con un subconjunto de ejercicios bastante adecuados para nuestro sistema mientras que el corpus de datos necesario para completar estos ejercicios estaba (en parte) disponible dentro de los tiempos requeridos por nuestro proyecto.

4.2.2. Modelo

Nuestro sistema resuelve dos problemas de QA similares pero distintos.

Por un lado, disponemos de una base de datos estructurada, con datos del área de la investigación y la producción en TICs en Argentina, que requiere un enfoque estructurado con métodos basados en ontologías y en formalizaciones de dominio. El enfoque aquí es traducir la pregunta formulada en lenguaje humano a un lenguaje más formal "comprensible"según el modelo de dominio.

Por otro lado, para evaluar métodos de QA para datos no estructurados, resolvimos algunos ejercicios de la competencia CLEF del '07. La base de conocimientos para estos ejercicios son algunos snapshots de Wikipedia en Español y en Inglés, anteriores al 2007. Sobre esta base de conocimiento, el enfoque no es "traducir"la pregunta a un lenguaje estructurado sino interpretarla y "compararla", mediante distintas métricas, con documentos y pasajes, buscando medidas estádisticas y otras condiciones que permitan *rankear* una respuesta candidata con un cierto grado de confianza, o bien determinar que no fue posible encontrar una respuesta para la pregunta.

Conceptualmente, el modelo consiste en los tres pasos típicos: la creación de la base de conocimientos optimizada, el análisis lingüístico de la pregunta y la generación de una respuesta desde la base de conocimientos optimizada a partir de la pregunta con sus anotaciones. Los pasos 1 y 2, la generación de la base de conocimientos optimizada y el procesamiento de la pregunta son procesos esencialmente análogos para la base de conocimientos estructurada y para la no estructurada. El tercer paso, la generación de respuestas, tiene distintos enfoques según el caso. En las siguientes secciones recorreremos ambos enfoques en conjunto, señalando las decisiones de diseño y los distintos módulos utilizados, haciendo las distinciones pertinentes cuando sea necesario.

En las secciones subsiguiente comentaremos las implementaciones de los distintos pasos, con sus decisiones de diseño y las tecnologías utilizadas.

4.3. Base de conocimiento

La creación de la base de conocimiento es el único que se ejecuta offline y consiste en la obtención del corpus original y en la generación de índices invertidos. Los índices invertidos, recordamos, son índices que permiten buscar, para una serie de términos vinculados lógicamente, un conjunto ponderado de documentos pertinentes. Por su naturaleza, este paso está separado de la ejecución del resto del código.

4.3.1. Corpus inicial

4.3.1.1. Estructurado

Los datos originales del proyecto mitic constan de una serie de documentos xml y de cinco colecciones de mongodb y relaciones entre ellas (una base de datos no relacional).

Las cinco coleccciones son: universidades, investigadores, empresas, publicaciones, proyectos y temáticas.

Estas colecciones definen el dominio. Cada entidad posee sus respectivos atributos y distintas relaciones con otras entidades. Estas relaciones contienen desde vínculos explicitos

como "trabajar-en"pero también relaciones inferidas mediante distintos algoritmos durante el proyecto mitic. Las relaciones que incorporamos tienen distintos pesos según este tipo de caracteristicas y tejen un grafo de distancias más o menos .[es]peculativas.[en]tre todas las entidades.

Los atributos de cada entidad se pueden ver en el Anexo: [[HACER Y LINKEAR]].

4.3.1.2. Wikipedia

El subset de ejercicios de CLEF'07 que elegimos resolver utiliza como corpus un snapshot de wikipedia en inglés y otro en español, ambos anteriores al año 2007. El set de preguntas completo involucra una serie de documentos para los cuales hay que registrarse en la asociación: algunas preguntas se responden en base a wikipedia y otras en base a estos documentos. Identificamos las preguntas que se responden desde wikpedia chequeando los documentos fuente de las respuestas esperadas.

Wikipedia ofrece diferentes formas para obtener una propia imagen de la enciclopedia. Es posible incorporar la base de datos de wikipedia a una instalación wikimedia propia, es posible descargar una imagen estática del sitio en archivos html y también se puede descargar un archivo xml gigante con todos los artículos.

En la guía para los participantes de la competencia [1] hay un link para descargar wikipedia en inglés preprocesada de Noviembre de 2006 [1] y dos opciones para descargar wikipedia en español: una imagen estática en html de Noviembre de 2006 [2] y un dump xml de Diciembre de 2006 [3]
[4]

La guía aclara que, bajo responsabilidad de los participantes, se puede usar cualquier otra versión de Wikipedia, siempre y cuando sea anterior a noviembre / diciembre de 2006. Además, se pide que las respuestas sean tomadas de "entradas reales.º artículos de wikipedia y no de otros documentos (por ejemplo: "image", "discussion", "category", etc).

Los links a páginas de wikipedia en español no estaban más disponibles (ambos respondieron 404), mientras que el formato preprocesado pedido para el inglés resultaba realmente complejo de instalar y parecía una linea muerta. Por estos motivos, seguimos la sugerencia sobre el uso responsable de otras wikipedias y utilizamos las siguientes:

Idioma	Fecha	Tamaño
Inglés [5]	4 de noviembre de 2006	7,6G
Español [6]	7 de julio de 2006	558M

El mantenimiento de un mismo formato para ambos idiomas nos permitió crear un indexador único para cualquier dump en xml de una wikipedia. Durante el desarrollo y las pruebas, para evitar tiempos de carga innecesarios y también para probar, utilizamos otras imagenes disponibles de wikipedia:

[1] `http://ilps.science.uva.nl/WikiXML/`
[2] `http://static.wikipedia.org/downloads/November_2006/es/`
[3] `http://download.wikimedia.org/images/archive/eswiki/20061202/pages-articles.xml.bz2`
[4] [[PEDIR WIKIPEDIA Y OTRA DB]]

Idioma	Fecha	Tamaño
Español	26 de enero de 2007	902M
Español	14 de junio de 2013	7,3G
Inglés simple	4 de julio de 2006	26M
Inglés simple	24 de julio de 2013	406M

4.3.2. Creación de Indices

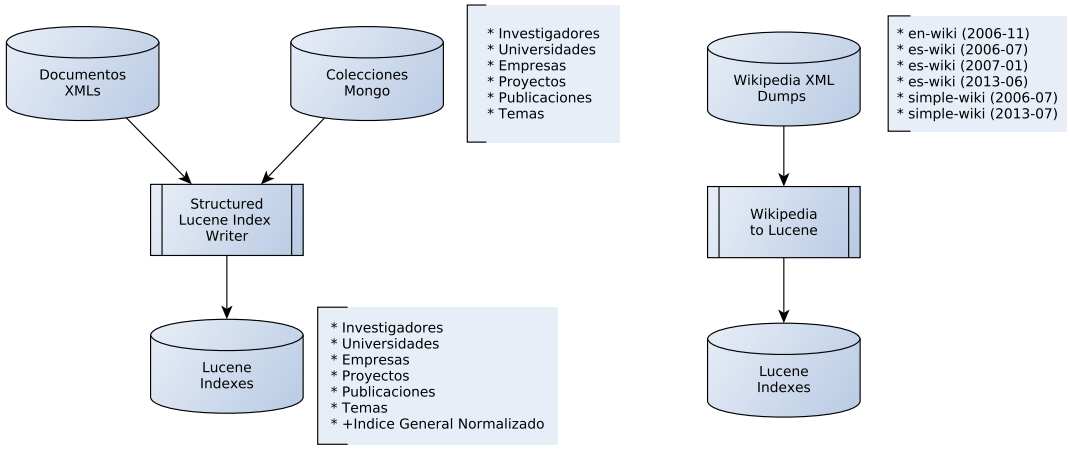

Fig. 4.1: Creación de Indices

A partir de esta base de datos de mongo y de los archivos xml fueron construidos cinco índices de búsqueda lucene y un índice de búsqueda más, general, con la información normalizada de los otros cinco. Cada uno de los cinco índices por entidad mantiene la estructura del tipo como campos de los documentos. Esto quiere decir que el índice invertido para *Investigadores* tiene los mismos campos del modelo de datos de nosql. Además, se agregó el campo "all"que resulta de la concatenación de todos los campos. Este campo resulta útil a la hora de filtrar resultados. El índice general posee un documento por cada entidad de las cinco colecciones, manteniendo también un puntero a la entidad original y su tipo.

Para la construcción de indices lucene con los dumps de wikipedia usamos la librería gwtwiki (Ver Apéndice A.9). Los artículos se indexan como documentos con los siguiente campos: *id, title, body y all*. En este proceso se descartan artículos mal formados y entradas representado imágenes o discusiones, tal como se sugiere en la guía. Por mera curiosidad, tomamos tiempos en la contrucción de estos índices locales sobre versiones de wikipedia. Estos son algunos tiempos de indexación para distintos dumps sobre una [[detalles de la compu]]:

Idioma	Tamaño	# Entradas	Tiempo
es	50M	# 16 millones	2.5min
es	50M	# 16 millones	2.5min
es	50M	# 16 millones	2.5min
es	50M	# 16 millones	2.5min

El proceso de creación de índices está ilustrado en la figura 4.1

4.3.3. Interfaz de servicios

La creación offline de índices lucene tiene como finalidad optimizar la base de conocimiento para responder con mayor eficiencia a búsquedas de resultados en un momento posterior. En esta sección vamos a ver la interfaz que presenta la base de conocimientos indexada al resto de los módulos del sistema y qué dependencias existen con los módulos de análisis lingüístico.

Para la base de conocimiento estructurada, reutilizamos un modelo de datos escrito en java del grafo de entidades que obtuvimos de los investigadores del proyecto mitic (Ver A.10). A estos modelos se les agregó soporte para su representación como documento dentro de un índice. Por ejemplo, el modelo para la entidad Üniversidad de Buenos Aires", un objeto de clase ÜniversidadNodo", además de persistirse en la colección UniversidadesGrafo de la base de datos de mongo también dispone de una representación como documento en un índice lucene particular UniversidadesIndex y otra en el índice general (GeneralIndex). A nivel colecciones, cada entidad dispone de un representante que maneja el acceso a su colección en la base de datos y también a su índice. A partir de estos representantes por entidad que ofrecen acceso a una base de datos y a un índice creamos la interfaz *KnowlegdeBase*. Las responsabilidades de esta interfaz se corresponden como un handler de conocimiento acerca de una cierta clase de entidades. En este punto del código reificamos entidades implicitas en el modelo. Estas entidades son, por ejemplo: Ciudad, Provincia, Centro de Investigación, etc. Entre otras funcionalidades, cada *KnowledgeBase* dispone de varios motores de generación de queries acordes al tipo de datos del índice. Otra de sus responsabilidades es verificar por la identidad de una entidad. Es decir, cada KnowledgeBase sabe responde, dado un string, si ese string *es* una entidad o no lo es, con un cierto grado de confianza.

[[Reificacion de entidades como algo]] [[Tablita con totales por entidad]] [[Relaciones

solo presentes en mongo]]

Las *KnowledgeBase* de las cinco entidad y el índice general están, a su vez, controlados por la interfaz *KnowlegdeManager* (KM). Las principales tareas de las que se hace cargo el KnowledgeManager en el proceso de generación de respuesta son:

- Verificar si un string se corresponde con una entidad (y de qué tipo)
- Verificar si un string está relacionado con el valor de algún atributo de alguna entidades (por ejemplo: si está presente en el campo especialidades de varias empresas)
- Verificar si un string es un nombre de campo (areas_inv,carreras, disciplina) o es un nombre de colección (universidades,empresas).
- Devolver una lista de documentos rankeados para una query (usando el índice general)

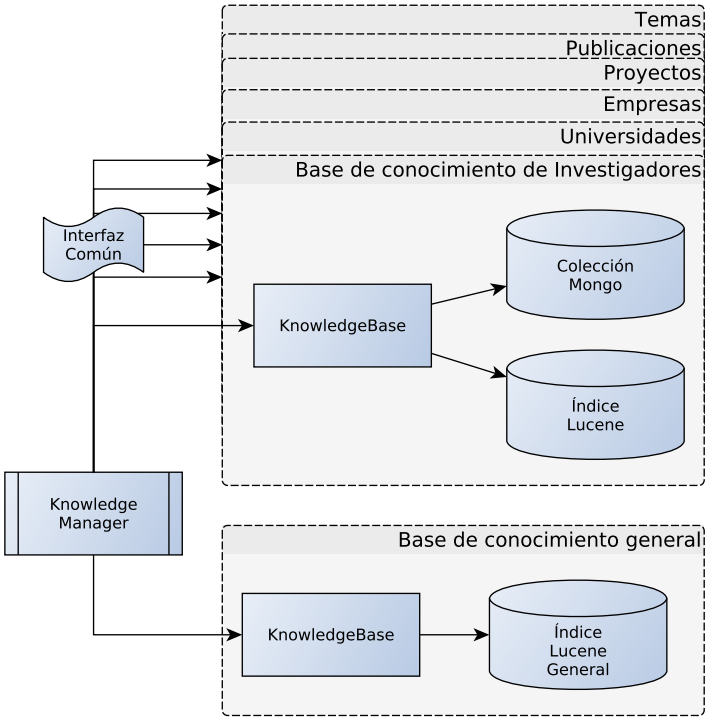

Fig. 4.2: Base de Conocimiento de grafo de tics

La base de conocimiento de los ejercicios de wikipedia es mucho más sencilla porque no existe ningún modelo *a priori* más allá del documento de lucene. El formato de la entidad "artículo", como señalamos antes, es: $(id, titulo, cuerpo)$. En este caso, el trabajo más fino no está en el modelado inicial del dominio sino la capacidad lingüística de extraer pasajes a partir de un artículo y recomponer información estructurada a partir de estos pasajes. Mientras que para un modelo estructurado la base de conocimiento debería permitirnos, para un cierto input, identificar univocamente una entidad y darnos pistas sobre un pedido de información acerca de esa entidad, el objetivo sobre un corpus de documentos en traer todos los documentos en los que sea posible que exista un pasaje respondiendo a la pregunta o evidencia relevante para apoyar una respuesta. Es decir, mientras una respuesta acotada es una virtud para el manejador de una base de conocimientos estructurada, el manejador de una lista de documentos de texto debería devolver una lista lo suficientemente grande para contener la respuesta dentro de los pasajes. La razón de esta política es que si por ser demasiado estrictos a la hora de retornar documentos llegasemos a descartar un pasaje candidato válido esto redundaría en una baja generar de efectividad, mientras que en pasos subsiguiente será trivial descartar toda información irrelevante sin tanto costo. Por eso, el acceso a los indices de wikpedia consta simplemente de un generador de queries similar al recién comentado accediendo y acumulando resultados (rankeados) a partir de un $LuceneIndexReader$ común (Ver A.8 para más información).

4.4. Análisis de la pregunta

El segundo paso comienza con la detección del idioma de la pregunta. A partir de esta detección se decide qué módulos se utilizarán para analizar la consulta y generar las anotaciones y descomposiciones en el resto del análisis.

En esta sección detallaremos el procesamiento realizado sobre la pregunta

QAnalyzer es la clase que implementa el análisis de la pregunta.
Este es el **algoritmo**:

1. Detectar idioma de la pregunta: los siguientes pasos priorizarán distintos analizadores de acuerdo al idioma reconocido.
2. Detectar NERs.
3. Detectar QWords
4. Detectar N-Gramas (1..n)
5. Detectar nombres de campos
6. Detectar nombres de colecciones
7. Detectar verbos
8. Eliminar palabras triviales

Los pasos 1 y 4 consisten en corroborar contra el KnowledgeManager los NERs que devuelven los analizadores y las distintos N-Gramas de 1 a n (actualmente n = 4).

Los pasos 5 y el 6 simplemente verifican igualdad en los sets de tokens (“universidades”, empresas”, etc)

Si en el paso 8 se procesaron menos del 80 % de los tokens de la palabra, el sistema se da
por vencido y devuelve una lista de score docs.
Si no, realiza un análisis estructurado basándose en la información obtenida.

4.4.1. Módulo de Detección de idiomas

Evaluamos distintas herramientas de detección de idiomas. Enumerar. Comentar cosas de otras herramientas. En general, los resultados son muy buenos. Casos borde donde pueden fallar o no, o donde el resultado no es obvio: “where is the universidad de buenos aires”. Este problema está particularmente presente en el procesamiento de preguntas, dado que son textos cortos en los que una construcción sustantivada en otro idioma puede desequilibrar erróneamente la balanza.

El módulo de detección de idiomas de nuestro sistema utiliza dos algoritmo de detección distintos, uno de la biblioteca Freeling y otro de Cybozu Labs, ambos escritos en java.

El detector de Cybozu está construido sobre modelos extraídos de las distintas wikipedias y posee una eficacia del 99 % para 49 idiomas, según presentación. Ciertamente, es un muy buen detector de idiomas. El detector de idiomas de freeling es así y asá.

Ambos permiten priorizar la detección de ciertos idiomas sobre otros. De esta manera podemos forzarlos a identificar sólo los idiomas esperados en nuestro dominio.

En nuestro sistema, configuramos ambos detectores para emitir un veredicto sólo en caso de confianza alta. El detector obtiene los resultados de ambos y desempata con un algoritmo ad-hoc en caso de resultados contradictorios.

4.4.2. Módulo de Traducción

Por el momento, no estamos utilizando ningún módulo de traducciones: todo el enfoque multilingüe está dado por la detección del idioma de la pregunta y la determinación de distintas herramientas de análisis según qué idioma sea. Sin embargo, en un momento se evaluó un enfoque distinto, basado en la traducción. Por ejemplo: utilizar módulos de procesamiento sólo en inglés y "normalizar" los inputs en otros idiomas (en principio, en español), a este idioma interno, y luego lo mismo con la generación de respuestas. A pesar de que no es el enfoque actual, hubo una fase de investigación dentro del dominio de la traducción, que resultó en un módulo de traducción basado en MyMemoryAPI.

Intento con google translator y la privatización. ¿La falta de software de traducción offline? El módulo de mymemory, robado de algún lugar. El sistema de cobro.

4.4.3. Comparadores

Dada la frecuencia en la que resultaba necesario comparar dos string, decidimos reificar la operación de comparación como una familia de clases que implementan Comparadores.

Gracias a esto se hace posible cambiar las nociones de igualdad o similaridad en un módulo completo del sistema o en alguna clase simplemente configurando otro comparador como parámetro. La interfaz permite a los distintos comparadores tomar valores binarios así como también valores entre 0 y 1. A su vez, esta considerada la posibilidad de configurar un umbral (threshold) a partir del cual redondear un valor entre 0 y 1 a un valor binario. Otro factor que tuvo mucha utilidad fue la capacidad de anidar comparadores. A nivel superclase también está permitida la posiblidad de ignorar o no ignorar la diferencia de mayúsculas, aunque este funcionamiento puede granularizarse en las instancias particulares. Los comparadores sirven, en general, para comparar tanto strings representando palabras como string representado listas de palabras (oraciones o textos). Algunos, en particular, sólo sirven para este segundo caso. Los comparadores disponibles actualmente son los siguientes, en un pseudo-orden decreciente de exactitud:

- Equal: compara por igualdad estricta
- EqualNoPunct: compara por igualdad, eliminando signos de puntuación y normalizando acentos y otras posibles diferencias que no deberían tenerse en cuenta.
- Contains: verifica si un string contiene a otro
- EditDistance: no implementado

Los siguiente comparadores son algoritmos fueron adaptados a partir de los Scorers del proyecto Qanus. Todos devuelven valores reales entre 0 y 1 y sirven para comparar textos (y no palabras). Estos comparadores, al igual que Contains, no son simétricos. Para distinguir, llamaremos primer string al buscado y segundo string a aquel en el cual se busca el primero.

- Frequency: cuenta la cantidad de veces que los tokens del primer string ocurren en el segundo string. Esta suma se divide por la longitud del segundo string, dando un valor entre 0 y 1.
- Coverage: cuenta cuantos tokens del primer string aparecen al menos una vez en el segundo, y divide esta suma por el total de tokens del *primer* string.
- Proximity: computa la distancia de

This feature computes the distance between the occurences of two search strings within a passage.

Typically the search string may span multiple word tokens in the passage.

To compute the distance, we will compute the midpoint of the span of the two search strings, and find the difference between the two midpoints.

* ... X .. X... X .... Y ... Y
* | |
* <----------->

* where X are matches to search string 1 and Y are matches to the other search string.

- TermSpan

* This feature makes use of the span of matching search terms within a passage.

* For example, suppose the passage has the following search term matches (denoted by X)

* ..... X ..... X ..... X .....
* .....a ..... b ..... c .....
*
* a, b, c are position of the matching term within the passage (in terms of words).
* The span is thus |c-a|.
*
* The score of this feature is given as
* # matching terms / span
*
* This score is guaranteed to be between 0 and 1 inclusive.

5.2.4.4 Módulos de procesamiento de lenguaje natural

Para el análisis lingüístico de la pregunta se utilizan distintas librerías disponibles, dependiendo del idioma detectado. Para idioma español, utilizamos la librería Freeling, que fue comentada anteriormente. De esta librería utilizamos el POS-tagger, el NER y el NEC (reconocimiento y clasificación de entidades nombradas, respectivamente). Para el inglés, utilizamos en parte los módulos de Freeling y en parte herramientas producidas por la universidad de Stanford, tomados de Qa-sys. El motivo por el que usamos dos librerías distintas es el esperado: los módulos de Stanford demostraron funcionar mucho mejor que los de Freeling para el inglés.

4.4.4. Tecnologías utilizadas

Stanford Question Classifier
Stanford POS
Stanford NER
Freeling POS
Freeling NER
QWords,
Verbos
Trabajo futuro: ¿Otros idiomas?

4.5. Generación de Respuestas

Una vez completada la generación de metadatos lingüísticos, se procede al tercer paso, que toma diferentes formas dependiendo de la información encontrada en el análisis de la pregunta.

Dependiendo de cuánto se haya procesado de la pregunta original, existen dos tipos de flujos distintos de generación de respuestas: el estructurado y el no estructurado.

La generación de respuestas estructurada se construye cuando la información obtenida en el análisis lingüístico es suficiente para generar un mapeo coherente a la ontología definida implícitamente por la base de conocimientos. El análisis no estructurado se ejecuta cuando la información generada no fue suficiente y consiste en intentar ubicar marcas que puedan reconducir al análisis al flujo estructurado (es decir, se intentan diferentes transformaciones a fin de identificar información que permita el análisis estructurado). Esto se realiza con métodos menos confiables y menos rápidos que en el análisis lingüístico inicial. Si el análisis falla en esta segunda barrera, se procede a dar una respuesta ad-hoc con la información disponible.

En flujo de respuesta estructurado depende de la cantidad de información que se haya identificado. Esta información, en este punto, tiene tres tipos: QType, NERs y verbos.

Como dice en [3] y también en [2]

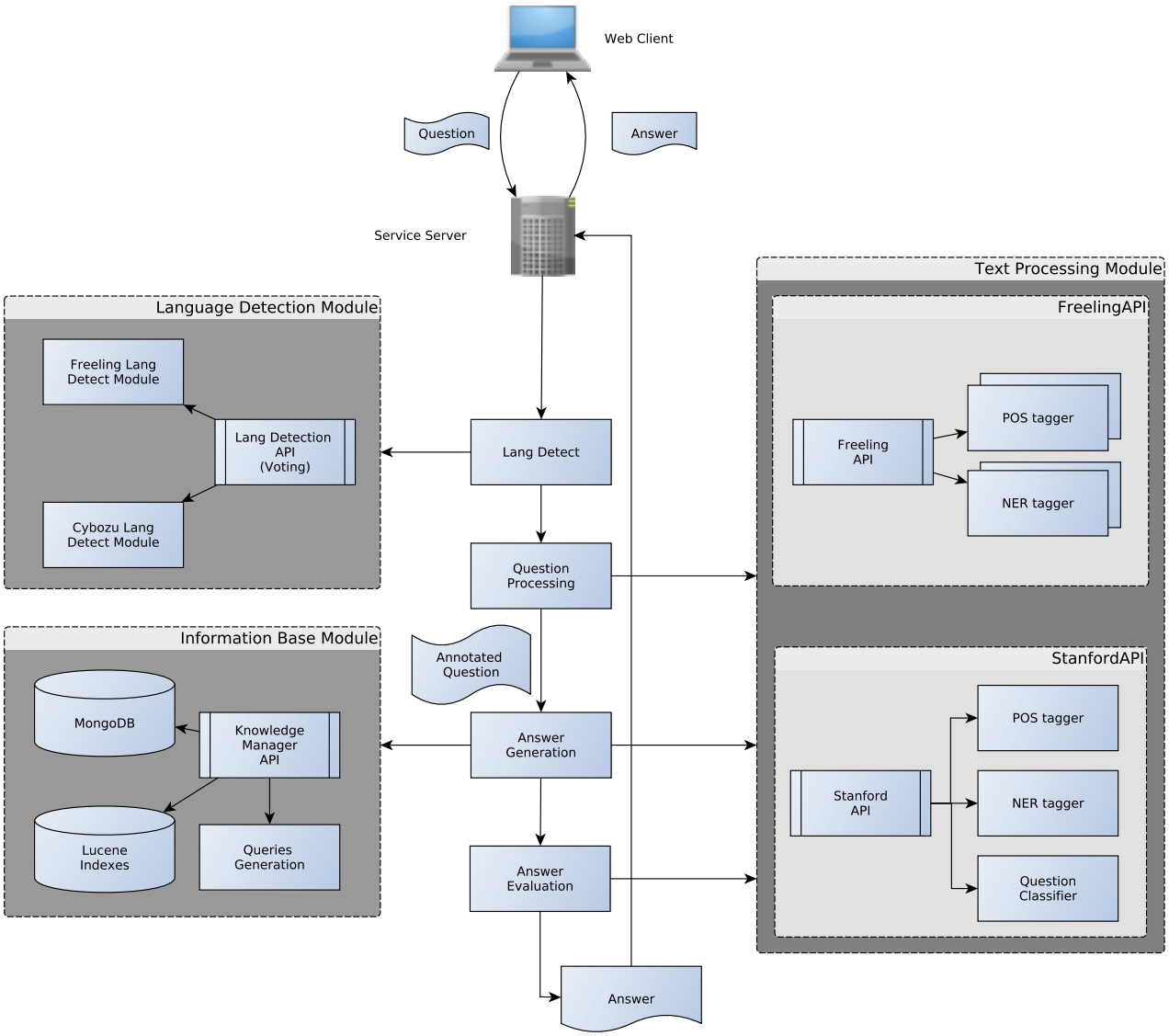

Fig. 4.3: Arquitectura

Apéndices

Apéndice A

HERRAMIENTAS

A.1. Stanford Question Classifier

The question classifier (QuestionClassifier) is implemented with the STANFORD CLASSIFIER (Manning and Klein 2003), using the question taxonomy explained in (Li and Roth 2002) and is responsible for finding out the expected answer type of a given question

Li, Xin, and Dan Roth. "Learning Question Classifiers.Ïnternational Conference on Computational Linguistics (COLING).2002. Manning, Christopher, and Dan Klein. .ºptimization, Maxent Models, and Conditional Estimation without Magic."Tutorial at HLT-NAACL and ACL. 2003.

This software is a Java implementation of a maximum entropy classifier. Maximum entropy models are otherwise known as conditional loglinear models, and are essentially equivalent to multiclass logistic regression models (though parameterized slightly differently, in a way that is advantageous with sparse explanatory feature vectors).

A.2. Stanford POS Tagger

Part of Speech Tagging Stanford POS Tagger (Toutanova and Manning 2000)

Toutanova, Kristina, and Christopher Manning. .ªnriching the Knowledge Sources Used in a Maximum Entropy Part-of-Speech Tagger."Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora. 2000. 63-70.

A.3. Stanford NER Tagger

Named Entity Recognition Stanford Named Entity Recognizer (Finkel, Grenager and Manning 2005)

Finkel, Jenny Rose, Trond Grenager, and Christopher Manning. Ïncorporating Non-local Information into Information Extraction Systems by Gibbs Sampling.roceedings of the 43nd Annual Meeting of the Association for Computational Linguistics. 2005.

A.4. Freeling

Freeling es una librería de código abierto que provee distintas herramientas de procesamiento de lenguaje natural, desarrollada y mantenida por el Centre de Tecnologies i Aplicacions del Llenguatge i la Parla (TALP) de la Universidad Politécnica de Cataluña (UPC). Freeling está diseñado para ser usada como una librería externa y ofrece un API en distintos lenguajes de programación. Su principal virtud es ser multilingüe, esto es, los

diferentes analizadores que provee funcionan para un conjunto bastante amplio de idiomas. La última versión a la fecha (3.1) soporta los siguientes idiomas:

- Asturian (as)
- Catalan (ca)
- English (en)
- French (fr)
- Galician (gl)
- Italian (it)
- Portuguese (pt)
- Russian (ru)
- Slovene (sl)
- Spanish (es)
- Welsh (cy)

Cabe destacar que no todos los módulos soportan todos los idiomas. Sin embargo, dado que el proyecto está radicado en España, los idiomas necesarios para los fines de nuestro trabajo (español e inglés), soportan todos los módulos disponibles en la librería. Freeling 3.1 ofrece los siguientes analizadores lingüisticos:

- Detección de idioma
- Tokenizer
- Sentence splitting,
- Análisis morfológico
- NER y NEC (Detección y Clasificación de Entidades Nombradas)
- Reconocimiento de fechas, números, magnitudes físicas, monedas
- Codificación fonética
- POS tagging,
- Shallow parsing
- Dependency parsing
- Wordnet-based sense annotation
- Word Sense Disambiguation
- Coreference resolution

A.5. Lang Detect de Cybozu Labs

Para la detección de idiomas utilizamos tanto el módulo que brinda Freeling como una librería de Cybozu Labs - una reconocida compañía japonesa, implementado en Java y liberado bajo Apache License 2.0. En la práctica, este paquete dio excelentes resultados. Soporta 53 idiomas con %99 de precisión para todos ellos (según sus tests). El detector se basa en perfiles de idiomas generados a partir de las distintas wikipedias y detecta el idioma de los texots usando un filtro bayesiano ingenuo (*naive bayesian*).

A.6. MyMemory

MyMemory es la Memoria de Traducción más grande del mundo: 300 millones de segmentos a finales de 2009

Como las MT tradicionales, MyMemory almacena segmentos con sus traducciones, ofreciendo a los traductores correspondencias y concordancias. El proyecto se diferencia de las tecnologías tradicionales por sus ambiciosas dimensiones y por su arquitectura centralizada y basada en la colaboración colectiva. Todos pueden consultar MyMemory o hacer aportaciones a través de Internet; la calidad de las aportaciones será cuidadosamente ponderada. MyMemory gives you quick access to a large number of translations originating from professional translators, LSPs, customers and multilingual web content. It uses a powerful matching algorithm to provide the best translations available for your source text. MyMemory currently contains 644.377.834 professionally translated segments.

Las memorias de traducción son almacenes compuestos de textos originales en una lengua alineados con su traducción en otras. Esta definición de memorias de traducción coincide literalmente con una de las definiciones más aceptadas de corpus lingüístico de tipo paralelo (Baker, 1995). Por esto se puede decir que las memorias de traducción son corpus paralelos.

A.7. Reverb

ReVerb is a program that automatically identifies and extracts binary relationships from English sentences. ReVerb is designed for Web-scale information extraction, where the target relations cannot be specified in advance and speed is important.

ReVerb takes raw text as input, and outputs (argument1, relation phrase, argument2) triples. For example, given the sentence "Bananas are an excellent source of potassium,- eVerb will extract the triple (bananas, be source of, potassium).

A.8. Apache Lucene

Lucene es una librería de information retrieval, de código abierto, escrita en Java y distribuida bajo la licencia Apache Software License por la Apache Software Foundation. No está pensada para usuarios finales sino para ser integrada dentro de proyectos informáticos, resolviendo la parte de bajo nivel y brindando servicios a través de un API en diferentes lenguajes de programación. Su core es un índice invertido como el que de-

scribimos anteriormente. La implementación de un sistema que utiliza Lucene consta de dos pasos separados:

- La **creación** del índice, es por lo general un proceso offline en el cual se incorporan distintas fuentes de información al índice
- La **búsqueda** de documentos en el índice creado en el paso anterior, a partir de una query ingresada por el usuario final. Este proceso se incorpora dentro del flujo 'online' del sistema. El resultado de esta búsqueda es una lista de documentos rankeados con un cierto puntaje.

Es importante señalar que si bien el proceso de creación del índice suele estar desacoplado del resto del sistema, las fuentes de información no tiene por que ser 'offline' en el sentido de ser documentos en un disco local. De hecho, Nutch, otro proyecto de código abierto de la Apache Software Foundation es un motor de búsqueda web basado en Lucene que incorpora un crawler para indexar sitios web. Lucene soporta cualquier fuente de información que pueda convertirse en texto mediante algoritmia.
Los conceptos fundamentales de Lucene son: índice, documento, campo, término y query.

- Un índice contiene un conjunto de documentos
- Un documento es un conjunto de campos
- Un campo es el nombre de una secuencia de términos
- Un término es un token (una palabra)
- Una query es una lista de términos conectados con distintos operados lógicos

[[Dar ejemplos de una query]]

A.9. gwtwiki -Java Wikipedia API

Java Wikipedia API (Bliki engine) http://code.google.com/p/gwtwiki/ Esta librería tiene métodos útiles para trabajar con dumps de wikipedia. La usamos para testear los métodos no estructurados de la tesis.

A.10. Modelos de Morphia

Java Wikipedia API (Bliki engine) http://code.google.com/p/gwtwiki/ Esta librería tiene métodos útiles para trabajar con dumps de wikipedia. La usamos para testear los métodos no estructurados de la tesis.

Bibliografía

[1] Clef 2007. Guidelines for participants qa@clef 2007. `http://celct.fbk.eu/QA4MRE/scripts/downloadFile.php?file=/websites/ResPubliQA/resources/past_campaigns/2007/Documentation/QA_2007_Track_Guidelines.pdf`.

[2] Bach and Badaskar. A review of relation extraction. *School of Computer Science*.

[3] George D. Greenwade. The Comprehensive Tex Archive Network (CTAN). *TUGBoat*, 14(3):342–351, 1993. Articulo de ejemplo.