PROCESAMIENTO DE LA GRAMÁTICA DE CONTEXTO DE UNA APLICACIÓN MÓVIL BASADA EN VOZ PARA CLIENTES ANDROID USANDO ONTOLOGÍAS DE DOMINIO

Diana Marcela Folleco Idrobo

Eliana Andrea Concha Agredo

Universidad del Cauca

Facultad de Ingeniería Electrónica y Telecomunicaciones

Departamento de Telemática

Línea de Investigación de Servicios Avanzados de Telecomunicaciones

Popayán, Marzo de 2012

PROCESAMIENTO DE LA GRAMÁTICA DE CONTEXTO DE UNA APLICACIÓN MÓVIL BASADA EN VOZ PARA CLIENTES ANDROID USANDO ONTOLOGÍAS DE DOMINIO

Diana Marcela Folleco Idrobo

Eliana Andrea Concha Agredo

Trabajo de grado presentado como requisito para optar al título de Ingeniero en Electrónica y Telecomunicaciones

Director

Francisco Orlando Martínez Pabón

Magister en Ingeniería, área Ingeniería Telemática

Universidad del Cauca

Facultad de Ingeniería Electrónica y Telecomunicaciones

Departamento de Telemática

Línea de Investigación de Servicios Avanzados de Telecomunicaciones

Popayán, Marzo de 2012

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

CAPÍTULO 1

INTRODUCCION.

1.1 PLANTEAMIENTO DEL PROBLEMA

La evolución de las comunicaciones inalámbricas, los sistemas distribuidos, el aumento de la potencia y la capacidad interactiva de los dispositivos de mano y portátiles, ofrecen a los usuarios la posibilidad de tener acceso amplio y continuo a los recursos informáticos en una amplia variedad de contextos [1]. Esto significa que gracias al crecimiento de las comunicaciones inalámbricas, la interacción humano-ordenador puede optimizarse [2].

Esta optimización puede utilizar una potente herramienta: la voz, el medio más importante de la comunicación humana. Los dispositivos móviles que incorporan aplicaciones de voz, pueden soportar una comunicación invisible, como una forma natural de comunicarse, y por consiguiente, se hace necesaria la utilización de técnicas de reconocimiento y síntesis de voz [3]. En este sentido, las aplicaciones basadas en dichas técnicas, pueden ayudar a los usuarios a centrarse en su trabajo actual, sin un esfuerzo adicional de las manos y los ojos [4] [5], brindando una verdadera interacción de usuario multimodal [6].

Entre las técnicas de reconocimiento y síntesis de voz más difundidas, se encuentran los motores TTS (Text to Speech) y ASR (Automatic Speech Recognition). En este sentido, existe una amplia gama de proyectos sobre motores que permiten el desarrollo de aplicaciones que incluyen la interacción humano-ordenador, pero la mayor parte del esfuerzo, se ha concentrado en la utilización de software propietario, lo cual ha llevado al desarrollo de aplicaciones móviles basadas en voz con gramáticas limitadas en dominios restringidos [7]. Por otro lado, aunque algunos motores Open Source presentan limitaciones asociadas a la baja calidad de sonido y la limitación de idiomas [8], se han definido algunos mecanismos para optimizar su funcionamiento, utilizando en muchos casos un servidor externo que realiza el procesamiento o post-procesamiento de síntesis de voz, según sea el caso [9].

De acuerdo a este contexto, es justificable que las aplicaciones de control basadas en comandos, sean hoy en día las más difundidas. Típicamente, este tipo de aplicativos ofrecen un vocabulario restringido que contiene los comandos admitidos. Este tipo de reconocimiento de voz es menos complejo que el reconocimiento de voz sin restricciones de lenguaje y por lo tanto se puede construir más fácilmente en dispositivos móviles [10].

En el mismo sentido, para los dispositivos móviles en particular, se impone una limitante adicional cuando se trata de incluir todo el vocabulario relacionado con el contexto de la aplicación, debido a las limitaciones intrínsecas de este tipo de terminales, relacionadas con la capacidad de procesamiento, memoria, batería y almacenamiento de datos. Por lo tanto, el éxito de la interacción con los sistemas de reconocimiento de voz depende en gran medida de la restricción del lenguaje o vocabulario específico a ser utilizado en un dominio de aplicación concreta [11] [12] y de la plataforma móvil empleada para esta tarea. Dicha restricción se utiliza para resolver la ambigüedad semántica del lenguaje natural, tal como se describe en [13] [14] [15].

En este sentido, el lenguaje que pueda ser reconocido por un reconocedor de voz o "vocabulario de entrada" se define como la gramática de contexto y se puede representar por medio de ontologías de dominio. En términos simples, una ontología de dominio define los términos abstractos, las clases de objetos y sus relaciones existentes en un contexto específico [16]. Las ontologías intentan formular un esquema conceptual exhaustivo y riguroso de un dominio dado, definiendo un vocabulario común que incluye además la interpretación de los conceptos básicos del dominio y sus relaciones, además permite realizar procesos de inferencia que conducen a la adquisición de nuevo conocimiento a partir del ya existente. En una aplicación móvil basada en voz, una ontología de dominio, representa el vocabulario de una gramática predefinida en una situación concreta. En esencia, el vocabulario se verá limitado a representar operaciones especializadas y gramáticas de dominio específico [17].

Por otro lado, aunque se han realizado algunos trabajos relacionados sobre sistemas operativos como iPhone, Symbian, Blackberry o Windows Phone [16], el tratamiento de aplicaciones móviles basadas en voz con especificación de gramáticas por medio de ontologías ha sido poco explorado sobre una de las plataformas más recientes en su incursión en el mundo móvil, como lo es Android [18]. Su amplia expectativa y crecimiento en los últimos años, así como también las grandes ventajas que ofrece a los desarrolladores de aplicaciones por sus características de código abierto y flexibilidad de configuración, son algunos de los factores que motivan la elección de esta plataforma para atender los requerimientos del presente trabajo de grado. De forma concreta, el presente proyecto pretende resolver el siguiente interrogante:

¿Cómo procesar la gramática de contexto de una aplicación móvil basada en voz usando ontologías de dominio para clientes Android?

1.2 OBJETIVOS DEL TRABAJO DE GRADO

1.2.1 Objetivo General

Definir un mecanismo que permita procesar la gramática de contexto de una aplicación móvil basada en voz para clientes Android usando ontologías de dominio.

1.2 2 Objetivos Específicos

- Determinar las capacidades de configuración y reconocimiento de voz sobre la plataforma Android para algunos motores ASR[1] de código abierto.
- Adaptar un mecanismo de razonamiento ontológico a partir de los ya existentes, que permita procesar la gramática de contexto de una aplicación móvil Android basada en voz.
- Construir un piloto que permita evaluar el funcionamiento del mecanismo, usando una ontología para representar la gramática en un dominio específico.

1.3 APORTES DEL TRABAJO DE GRADO

El presente trabajo de grado pretende proporcionar los siguientes aportes investigativos:

- Evaluación de las capacidades de configuración y reconocimiento de voz sobre la plataforma Android de algunos motores ASR de código abierto.

- Adaptación de un mecanismo de razonamiento ontológico para procesar la gramática de una aplicación móvil Android basada en voz a través de la exploración de métodos de razonamiento de ontologías existentes.
- Piloto de una aplicación móvil basada en voz para un contexto específico, que evalúa la funcionalidad del mecanismo de procesamiento de gramática de contexto.

1.4 ESTRUCTURA DEL TRABAJO DE GRADO

De acuerdo al contexto planteando, se han definido las siguientes secciones:

Capítulo 2: se abordan las definiciones formales de conceptos claves para el entendimiento del proyecto realizado; se construye una base inicial de conocimiento sobre los temas directamente relacionados con el presente trabajado de grado, en la cual se incluyen sus características más relevantes y las definiciones utilizadas para el mismo.

Capítulo 3: se presenta una evaluación de los módulos requeridos para el procesamiento de la gramática de contexto de una aplicación móvil basada en voz para clientes Android usando ontologías de dominio, determinando cuáles son los motores ASR Open Source que presentan mejores capacidades de configuración sobre la plataforma Android, y qué mecanismo de razonamiento ontológico ofrece un mejor procesamiento de dicha gramática, por medio de la especificación algunos criterios de evaluación para realizar su comparación con las herramientas software que implementan tales módulos.

Capítulo 4: se define el mecanismo de procesamiento de la gramática de contexto de una aplicación móvil Android basada en voz, a través del estudio, definición e integración de los módulos involucrados: los motores ASR de libre distribución y los componentes de razonamiento ontológico, y para éstos últimos se adapta un mecanismo de razonamiento a partir de los existentes. Adicionalmente, se define la gramática de contexto representada por la ontología de dominio elegida, se plantea una arquitectura base para el procesamiento de la misma, y finalmente, se describe un piloto con una aplicación móvil basada en consultas por voz para un contexto específico, para evaluar diferentes métricas de reconocimiento de voz y procesamiento del mecanismo creado.

Capítulo 5: se resumen los aportes y conclusiones del trabajo y se plantean futuros proyectos relacionados.

Anexo A: Esquema para evaluar ontologías únicas para un dominio de conocimiento.

Anexo B: Análisis y Diseño del Piloto de prueba.

Anexo C: Manuales de Instalación y Configuración de todas las herramientas empleadas para el reconocimiento de voz y razonamiento ontológico.

CAPITULO 2

ESTADO DEL ARTE

MARCO TEORICO

En el presente capítulo se realiza una contextualización de los términos más relevantes para el presente trabajo de grado y se presentan algunos de los trabajos relacionados , identificando sus brechas y aportes a la presente investigación.

2.1 CONTEXTUALIZACION

Esta sección contiene los conceptos más relevantes que permitirán la comprensión del entorno en el que se desarrolla el problema central del presente trabajo de grado. A continuación se explican conceptos relacionados con las aplicaciones móviles basadas en voz, la gramática de contexto, los sistemas de reconocimiento de voz y el razonamiento de ontologías de dominio.

2.1.1 Aplicación móvil basada en Voz

Las aplicaciones de voz en computación móvil, normalmente permiten al usuario controlar vocalmente las funcionalidades del dispositivo, una característica conocida como comando y control [19]. En este sentido, el uso del reconocimiento de voz permite el control de los dispositivos de comunicaciones con las manos libres prácticamente hablando, éste control incluye sistemas de música, sistemas de navegación, búsqueda por voz, entre otros [20].

Hoy en día, las aplicaciones de voz tienden a concentrarse en los servicios básicos, tales como la navegación, la marcación, mensajería y búsqueda [20] y los dispositivos como los smartphones permiten controlar varios aspectos con la voz: dialer de voz, lector de pantalla, lectura y dictado de correos electrónicos y SMS, lectura de la identificación de la llamada entrante, aplicaciones de navegación, informaciones geo-localizadas. Los smartphones de nueva generación han estimulado el nacimiento de una gran cantidad de aplicaciones que utilizan las capacidades de voces sintetizadas y de reconocimiento de voz, entre las cuales se encuentran: cursos de idiomas, traductores, juegos y demás, controlados a su vez por medio de interfaces de voz.

2.1.2 Sistemas de reconocimiento de Voz

Los sistemas de reconocimiento de voz se introdujeron por primera vez en la década de 1940 y han mejorado constantemente desde entonces [21]. En este sentido, los ASR (Automatic Speech Recognition) pueden ser definidos como la trascripción del lenguaje hablado por un sistema computarizado a texto leíble en tiempo real, independientemente del dispositivo que se utiliza para registrar la voz (un transductor, un micrófono, entre otros) [22].

El proceso de reconocimiento de voz puede ser visto como una secuencia de pequeños subprocesos, en una primera etapa, la entrada se registra y se divide en una representación espectral, que muestra las frecuencias para cada sonido. Esto se puede lograr por ejemplo con una Transformada Rápida de Fourier (FFT), con una frecuencia de muestreo de 100Hz. Estos fragmentos, son representados por

vectores que sirven como entrada para un Modelo Oculto de Markov (HMM) el cual, se utiliza para determinar la secuencia más probable de fonemas. Durante el proceso de reconocimiento de voz, las secuencias de fonemas son usadas como entrada y la palabra más probable para esta secuencia es devuelta [23-26]. Una palabra o secuencia de palabras obtenidas de los cálculos de los HMM es después pasada a un modelo de lenguaje para su posterior procesamiento.

La siguiente figura muestra un diagrama en bloques de un reconocedor de voz:

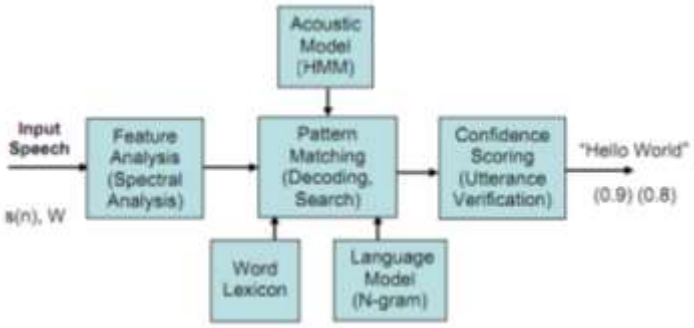

Figura 1. Diagrama en bloques de un Sistema ASR

El reconocedor se compone de tres etapas de procesamiento, "Feature Analysis" (Análisis de características), "Pattern Matching" (Coincidencia de Patrones) y "Confidence Scoring" (Puntuación de Confianza), junto con tres bases de datos: el conjunto de modelos acústicos, el léxico de palabras y el modelo de lenguaje. El objetivo del módulo "Feature Analysis" es extraer el conjunto de características más destacadas que representan las propiedades espectrales de los distintos sonidos del habla (las sub-unidades de palabras). El trabajo del módulo "Pattern Matching" es la combinación de la información (probabilidades) del modelo acústico, el modelo de lenguaje y el léxico, para encontrar la secuencia "optima", es decir, la secuencia de palabras que es consistente con el modelo de lenguaje y tiene la probabilidad más alta entre todas las secuencias de palabras posibles en el idioma. Finalmente, el objetivo del módulo "Confidence Scoring" es identificar posibles errores de reconocimiento como los eventos fuera del vocabulario y de ese modo mejorar potencialmente el rendimiento del algoritmo de reconocimiento [22].

2.1.2.1 Modelos Acústicos

El objetivo de los modelos acústicos es caracterizar la variabilidad estadística del conjunto para cada uno de los sonidos básicos (o palabras) del lenguaje. El método estadístico HMM [27-30] se utiliza para modelar la variabilidad espectral de cada uno de los sonidos básicos del idioma con una distribución de densidad mixta Gaussiana [31, 32], que es perfectamente alineado con un conjunto entrenado del habla, y es actualizado y mejorado de forma iterativa hasta una alineación optima [22]. Un modelo acústico es creado mediante la adopción de grabaciones de voz y sus transcripciones, para después compilarlos en representaciones estadísticas de los sonidos de las palabras [33].

2.1.2.2 Léxico de palabras

El propósito del léxico o diccionario de palabras es definir el rango de la pronunciación de las palabras en el vocabulario de trabajo [26, 34]. La razón de la necesidad de tal léxico de palabras, es que la misma ortografía puede pronunciarse de manera diferente por personas con distintos acentos, o porque las palabras tienen múltiples significados que cambian la pronunciación por el contexto que se esté usando. Un ejemplo de la variabilidad en la pronunciación de la ortografía es la palabra "record" que puede ser una marca para un jugador, o el proceso de grabar un sonido, es por esto que los diferentes significados tienen significativamente pronunciaciones muy diferentes [22].

2.1.2.3 Modelo de Lenguaje (LM)

El propósito del modelo de lenguaje [35, 36], es proporcionar una tarea de sintaxis que define las frases habladas de entrada que son aceptables y permite calcular la probabilidad de una cadena de palabras dado el modelo de lenguaje [22]. Siendo éste, uno de los pasos importantes de los sistemas de reconocimiento de voz, ya que es responsable de asignar una cadena de palabras a una secuencia de fonemas.

La elección de un modelo de lenguaje para un sistema de reconocimiento de voz depende principalmente de la finalidad del sistema, y existen dos enfoques principales que se utilizan en la mayoría de los sistemas de reconocimiento de voz, estos son:

1. **Enfoque estadístico usando modelos N-Gram.**

El método estadístico se basa en un corpus grande de texto para predecir la secuencia de palabras más probable o la palabra más propensa a seguir una cierta secuencia. La predicción se basa en las probabilidades de cadenas de palabras extraídas del corpus, llamadas N-grams. El "N" describe la longitud de la secuencia, donde se llama a una palabra unigram, dos palabras bigram (o 2-gram) y así sucesivamente. Los N-grams más utilizados en los modelos de lenguaje son bi y tri-grams, ya que la misma secuencia de dos o tres palabras es más probable que ocurra varias veces en un texto, que para n>3 [21].

2. **Gramáticas**

Las gramáticas se utilizan principalmente para los reconocedores de voz en aplicaciones de sistemas de diálogo y con ellas, se puede especificar explícitamente sentencias para el reconocedor de voz. A diferencia de un sistema de dictado, el conjunto de comandos y expresiones es finito y no una gran variedad de frases para reconocer.

Adicionalmente, el uso de gramáticas y un formato adecuado, permiten la integración de la comprensión del lenguaje natural, el cual es necesario para que la gramática esté integrada en el sistema de diálogo. Por este motivo, los comandos pronunciados por un usuario tienen que estar vinculados a una representación que el sistema pueda procesar [21].

El procedimiento habitual para la escritura de una gramática de reconocimiento de voz, es analizar las sentencias que puedan ser producidas. En consecuencia, una palabra o expresión que no es parte de las sentencias que se definen, no es reconocida por la gramática.

2.1.2.4 Parámetros de entrada de audio

El audio es en sí un fenómeno análogo, y la grabación de una muestra digital se realiza mediante la conversión de la señal analógica del dispositivo que se emplea para registrar la voz, en una señal digital a través del convertidor A/D. Básicamente, el convertidor A/D registra el valor de la tensión eléctrica registrada a intervalos específicos.

Hay tres factores importantes durante este proceso. El primero es la "frecuencia de muestreo", o la frecuencia con que son registrados los valores de tensión. En segundo lugar, están los "bits por muestra", o la precisión del valor registrado. Un tercer elemento es el número de canales (monofónico o estéreo), y para la mayoría de las aplicaciones ASR, una configuración en “monofónico” es suficiente. La mayoría de las aplicaciones, usan valores pre-establecidos de estos parámetros y el usuario no los debe cambiar a menos que la documentación lo sugiera. Los desarrolladores pueden experimentar con diferentes valores para determinar los que mejor funcionan en sus algoritmos.

Típicamente, el ancho de banda del habla es relativamente bajo (entre 100Hz-8kHz), esto significa que 8000 muestras/s (8 kHz) son suficientes para aplicaciones ASR más básicas, sin embargo algunas personas prefieren 16.000 muestras/s (16 kHz), ya que proporciona información de alta frecuencia más precisa, y éstas se pueden usar si se tiene suficiente procesamiento. Para la mayoría de aplicaciones ASR emplear frecuencias de muestreo superiores a los 22 kHz es un desperdicio.

2.1.2.5 ASR de código abierto

1. **API de Reconocimiento de voz de Android**

Android de Google es el sistema operativo para teléfonos móviles de código abierto y está disponible bajo una licencia libre que permite a los desarrolladores usar y modificar el código que ofrece Google a través de una Open Handset Alliance. La Open Handset Alliance tiene disponible el reconocimiento de voz básico como parte del paquete de código abierto, este está incluido en forma nativa, ya que el SDK hace que sea fácil de integrar la entrada de voz directamente en una aplicación móvil, por medio de una API que consta de una interfaz y varias clases que permiten acceder al servicio de reconocimiento de voz de Google [18].

Android como una plataforma abierta puede hacer uso de cualquier servicio de reconocimiento de voz registrado en el dispositivo, asi por ejemplo, la aplicación Google's Voice Search, que viene preinstalada en muchos dispositivos Android, mostra un botón “Speak now”, que al ser presionado envía el audio a los servidores de Google, los cuales soportan varios idiomas para la entrada de voz [18].

Aunque la integración de la entrada de voz en la búsqueda de Google es un paso importante hacia un mayor acceso ubicuo, plantea muchos problemas en términos de rendimiento del núcleo de las tecnologías del habla y el diseño de interfaces de usuario efectivas [18].

A diferencia de GOOG-411 [44],, que es un servicio beta dependiente del dominio que sirve para poder efectuar búsquedas vocales en el grande banco de datos de Google, la búsqueda por voz de Google debe ser capaz de manejar cualquier cosa que la búsqueda web de Google pueda manejar. Esto hace que sea un problema de reconocimiento desafiante, ya que el vocabulario y la complejidad de las consultas son demasiado grandes [18].

La siguiente figura muestra la arquitectura básica del sistema del reconocedor detrás de la búsqueda de Google por voz.

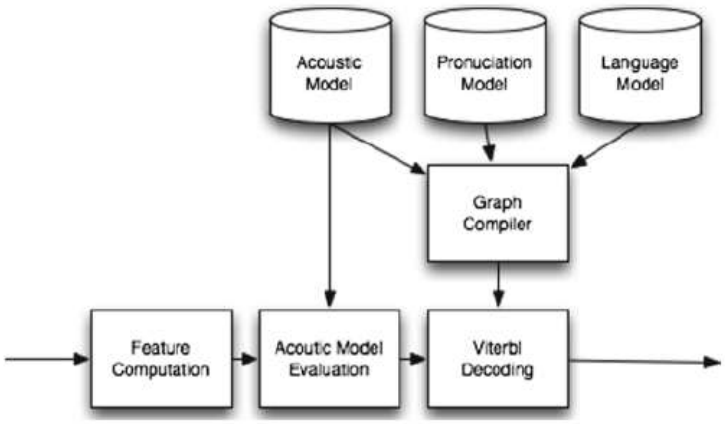

Figura 2. Diagrama en bloques básicos del reconocedor de Google

El objetivo del modelo de lenguaje es modelar la distribución probabilística desconocida *p(x)* de las secuencias de palabras en el idioma. Los modelos acústicos proporcionan una estimación de la probabilidad de las características observadas en un marco de la voz dado un contexto fonético particular [20].

2. Julius

Julius un software de reconocimiento de voz de código abierto para vocabulario largo y continuo usado en aplicaciones industriales y de investigación académica. Éste ejecuta reconocimiento de voz en tiempo real de una tarea de dictado de 60k palabras con tamaño reducido, e incluso en dispositivos embebidos [37]. También tiene gran versatilidad y escalabilidad, se puede construir fácilmente un sistema de reconocimiento de voz mediante la combinación de un modelo de lenguaje (LM) y un modelo acústico (AM). Julius está escrito en lenguaje C puro y se ejecuta en Linux, Windows, Mac OS X, Solaris y otras variantes de Unix. Ha sido portado al microprocesador SH-4A [38], y también se ejecuta en el iPhone de Apple [39]. La mayoría de los institutos de investigación en Japón utiliza Julius para su investigación [40, 41], y se ha aplicado en varios idiomas, como Inglés, Francés [42], el Chino Mandarín [43], Tailandés [44], Estonio [45], Esloveno [46] y Coreano [47]. La licencia es

similar a la licencia BSD, y no se imponen restricciones para la investigación, el desarrollo o incluso propósitos comerciales. Una visión general de Julius se ilustra en la Figura 3 [48].

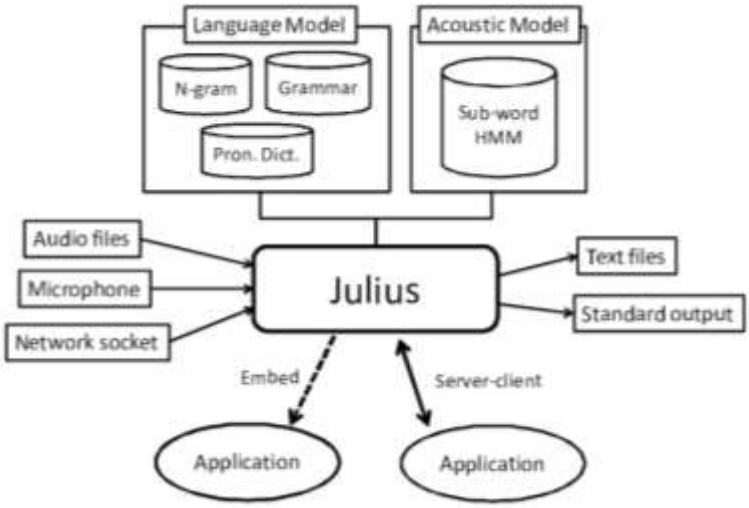

Figura 3. Visión general de Julius

Teniendo en cuenta un modelo de lenguaje y un modelo acústico, Julius funciona como un sistema de reconocimiento de voz. El modelo de lenguaje se compone de un diccionario de pronunciación de las palabras y una restricción sintáctica. Los tipos de modelo de lenguaje compatibles son: el modelo de palabras N-gramm, gramáticas basadas en reglas y una lista de palabras simples para el reconocimiento de palabras aisladas. Los modelos acústicos deben ser HMM definidos para las sub-unidades de palabras. Las aplicaciones pueden interactuar con Julius de dos maneras: en primer lugar, tipo cliente-servidor basado en sockets, y en segundo lugar, empleando la librería embebida. En cualquier caso, el resultado se incorporará a la aplicación tan pronto como finaliza el proceso de reconocimiento de una entrada de voz [48].

Las especificaciones de las características acústicas, el modelo acústico, los modelo de lenguaje y diccionario soportados por Julius son descritos a continuación:

A. Modelo Acústico.

El Modelo acústico (AM) de Julius soporta HMMs monofónico y trifono con cualquier número de mezclas, estados y unidades de fonemas [49]. Actualmente Julius también soporta HMM multi-stream, y MSD-HMM [50] grabado por HTS [51]. La función de probabilidad de salida debe ser una mezcla de densidades gaussianas con covarianza diagonal, mientras que la adaptación del hablante y los modelos de covarianza completa aún no son compatibles. El formato del archivo debe estar en formato ASCII HTK, y éste puede ser convertido al archivo binario Julius.

B. Diccionario

El formato del diccionario de pronunciación es común a todos los tipos de LM y se basa en el formato de diccionario de HTK (Hidden Markov Model Toolkit) [52] . Cada pronunciación debe ser una

secuencia de nombres de las unidades sub-ordenadas, tal como estén definidas en el modelo acústico, así mismo múltiples pronunciaciones de una palabra se pueden especificar como entradas separadas.

C. Modelo de Lenguaje

Julius admite reconocimiento de voz basado en modelos N-gram, gramáticas basadas en reglas y un diccionario solo.

N-gram: las longitudes arbitrarias de N-gram son compatibles con la versión más reciente, también soporta Class N-gram, en cuyo caso las probabilidades de la palabra in-class, no debe realizarse en el modelo de lenguaje, sino en el diccionario. Se debe tener en cuenta que las versiones anteriores a (3.x) son compatibles solamente con 3-gram o inferiores. El formato del archivo debe estar en el formato estándar ARPA, el más popular y soportado por varias herramientas de modelo de lenguaje. Como el archivo ARPA es un formato basado en texto y necesita mucho tiempo para ser analizado, Julius proporciona una herramienta "mkbingram" para convertirlo en el formato binario pre-compilado.

Gramática basada en reglas: Julius puede llevar a cabo el reconocimiento de voz basado en una gramática escrita. El formato del archivo está basado en una forma BNF-like en dos archivos separados, en el primero, la sintaxis está escrita a nivel de categoría y en el segundo, se escribe la entrada léxica por cada categoría. Julius puede utilizar múltiples gramáticas a la vez, en este caso se elige la mejor hipótesis entre todas las gramáticas.

Reconocimiento de palabras aisladas: Cuando se da sólo un diccionario sin ningún tipo de modelo de lenguaje, Julius llevará a cabo el reconocimiento de palabras aisladas. Dado que el proceso de reconocimiento no requiere de aproximación entre palabras, el proceso se termina en el primer paso.

3. PocketSphinx

CMU Sphinx es uno de los sistemas de código abierto más populares de reconocimiento de voz. Actualmente es utilizado por los investigadores y desarrolladores en muchos lugares en todo el mundo, incluyendo universidades, institutos de investigación y en la industria. Los términos liberales de la licencia de CMU Sphinx lo han convertido en un miembro importante de la comunidad de código abierto y ha proporcionado una forma económica de crear negocios alrededor del reconocimiento de voz para las empresas [37].

PocketSphinx es un pequeño sistema de reconocimiento continuo de voz, distribuido libremente bajo una licencia BSD simplificada, adecuado para aplicaciones portátiles y de escritorio. Cuenta con [53]:

- Múltiples plataformas: Linux, Windows, Mac OS X, iPhoneOS
- Soporte experimental para Nokia S60v3 y Windows Mobile
- Soporte a los modelos acústicos semi-continuos, fonéticamente-ligados y completos
- Modelos de lenguaje trigram y gramáticas de estados finitos JSGF.
- Modelos acústicos y pequeños modelos de lenguaje para Inglés y Mandarín

De acuerdo con la estructura del habla, los tres modelos que se utilizan en el reconocimiento de voz, se combinan para reconocer el habla, y se encuentran disponibles para su descarga [54].Estos son:

1. Modelo acústico que contiene las propiedades acústicas de cada sonido. Existen modelos independientes del contexto que contienen las propiedades (característica más probable de los vectores para cada fonema), y dependientes del contexto (construido a partir de sonidos del contexto) [54]. Todos los modelos utilizan el nuevo "40-phone" y son compatibles con CMUDict 0.6d y 0.7, y con la Herramienta Base de Conocimiento Sphinx. Estos modelos son liberados bajo la misma licencia permisiva de Sphinx-3 y se empaquetan para su uso con Sphinx-3 y PocketSphinx [55].

2. Diccionario fonético que contiene un mapeo de las palabras a fonemas [54]. El decodificador tiene que saber la pronunciación de las palabras, y por esto, el archivo de diccionario (frecuentemente con extensión .dic) es una lista de palabras con la secuencia de fonemas. En el caso en que una palabra pueda ser pronunciada de más de una forma, hay que proporcionarle un nombre distinto. De igual forma, se puede emplear un diccionario noise-word (noisedict), el cual es un conjunto de palabras compuestas por los fonemas de ruido. Un diccionario noise-word no es necesario, pero cuando se utiliza, puede mejorar significativamente la precisión en el reconocimiento gracias al manejo de secciones no verbales de la entrada [56].

3. Modelo de lenguaje se usa para restringir la búsqueda de palabras. Define la palabra que podría seguir a la previamente reconocida y ayuda a restringir de manera significativa el proceso de comparación de palabras extraídas improbables [54]. PocketSphinx utiliza los dos tipos de modelos que describen el lenguaje, las gramáticas que describen tipos muy simples de lenguaje para comando y control, por lo general escritas manualmente en formato JSGF o generados automáticamente con el código de formato [57], y los modelos de lenguaje estadísticos en un archivo LM (frecuentemente con la extensión .lm).

- ✓ **Formato de gramática JSGF (JSpeechGrammarFormat).**

Es una plataforma de representación textual de gramáticas para su uso en el reconocimiento de voz independiente del proveedor, y es utilizada para determinar lo que el reconocedor debe escuchar, y así describir las declaraciones que un usuario puede decir. JSGF adopta el estilo y las convenciones del lenguaje de programación Java™. Se utiliza una representación textual que es legible y editable por los desarrolladores y equipos, y se puede incluir en el código fuente [58].

Una regla gramatical especifica los tipos de expresiones que un usuario puede decir (una palabra hablada es similar a una frase escrita), por ejemplo, una gramática simple de control puede escuchar "open file", "close window", y comandos similares. Las reglas gramaticales se encargan de proporcionar un reconocedor de voz la gramática apropiada.

Una gramática se compone de un conjunto de reglas que definen conjuntamente lo que se puede hablar. La parte de una gramática que define exactamente lo que puede hablar un usuario, es denominada “token”.

El formato de gramática JSpeech permite gramáticas en varios idiomas. Sin embargo, la mayoría de los reconocedores operan mono-lingüísticos para una gramática típica que contiene solamente un

idioma. La mayoría de los reconocedores tiene un vocabulario amplio para cada idioma que soportan, sin embargo, nunca es posible incluir el 100% de un idioma.

Un solo archivo define una gramática única, por tanto, la definición de la gramática consta de dos partes: la cabecera y el cuerpo. La cabecera incluye una cabecera de auto-identificación, declarando el nombre de la gramática y las importaciones de las reglas de otras gramáticas. El cuerpo define las reglas y algunas pueden ser públicas y, la cabecera de la gramática puede incluir opcionalmente declaraciones *import*, las cuales permiten que una o todas las reglas públicas de otra gramática puedan ser referenciadas localmente.

Los dos patrones de definiciones de reglas son los siguientes:
<ruleName> = ruleExpansion;
public <ruleName> = ruleExpansion;

Los componentes de la definición de la regla son (1) una declaración pública opcional, (2) el nombre de la regla que se está definiendo, (3) un signo igual '=', (4) la expansión de la regla, y (5) cierre con punto y coma ';'.

PocketSphinx utiliza el formato de gramática JSGF, para describir

A. Expansiones de reglas.

Las expansiones de reglas más simples son una referencia a un token y una referencia a una regla. Por ejemplo, <a> = elephant; <b> = <x>;

La regla <a> se expande al token "elephant". Por lo tanto, para hablar <a> el usuario debe decir la palabra "elephant". La regla <b> se expande a <x>. Esto significa que para hablar <b>, el usuario debe decir algo que coincide con regla <x>. Por otro lado se pueden definir reglas de gramática complejas por combinaciones lógicas de las expansiones legales mediante:

- *Composition*: secuencias de ampliaciones y conjuntos de alternativas de expansiones.
- *Grouping* el uso de paréntesis y corchetes.
- *Unary Operators* para la repetición de las expansiones.
- *Attachment* de las etiquetas específicas de la aplicación para las expansiones.

B. Expansiones de reglas

- *Secuencias.*

Una regla puede ser definida por una secuencia de expansiones, cada una separada por un espacio en blanco. Las siguientes son definiciones de regla legales:

<where> = I live in Boston;
<statement> = this <object> is <OK>;

Para hablar una secuencia, cada elemento de la secuencia debe ser hablado en el orden definido. En el primer ejemplo, decir la regla <where>, el hablante debe decir las palabras " I live in Boston", en ese

orden exacto. En el segundo ejemplo se mezclan tokens con referencias a las reglas <object> y <OK>.Para decir la regla <statement>, el usuario debe decir "this", seguido de algo que coincide con <object>, después "is", y finalmente algo que concuerde con <OK>.

- *Alternativas.*

Una regla puede ser definida como un conjunto de extensiones alternativas separadas por caracteres de barra vertical `| 'y, opcionalmente, por espacios en blanco. Por ejemplo: <name> = Michael | Yuriko | Mary | Duke | <otherNames>;
Al decir la regla <name>, el hablante debe decir únicamente uno de los elementos del conjunto de alternativas. Por ejemplo, un hablante puede decir "Michael", "Yuriko", "Mary", "Duke" o cualquier otra palabra que coincida con la regla <otherNames>. Sin embargo, el hablante no puede decir "Mary Duke".

- *Pesos.*

No todas las formas de hablar una gramática son igualmente probables. Los pesos se pueden unir a los elementos de un conjunto de alternativas para indicar la probabilidad de cada alternativa hablada. Un peso es un numero punto flotante rodeado de shlashes, y cuanto mayor sea el peso, es más probable es que una entrada se hable. Por ejemplo:

<size> = /10/ small | /2/ medium | /1/ large;

Los pesos deben reflejar los patrones de ocurrencia de los elementos de un conjunto de alternativas. En el primer ejemplo, el escritor de la gramática indica que es "small" es 10 veces más probable que se hable, que "large" y tiene 5 veces más probabilidades que "medium".

C. Grouping

- *(Paréntesis).*

Cualquier expansión legal puede ser agrupada de forma explícita usando paréntesis coincidentes '()'. Grouping tiene alta prioridad, por lo tanto se puede utilizar para garantizar la correcta interpretación de las reglas, lo cual lo hace útil para mejorar la claridad. Por ejemplo, ya que las secuencias tienen mayor prioridad que otras alternativas, los paréntesis son necesarios en la definición de la siguiente regla para que "please close" y "please delete" sean legales.
<action> = please (open | close | delete);

- *[Agrupación opcional].*

Los corchetes se pueden colocar alrededor de cualquier definición de reglas que indica que los contenidos son opcionales. En otros aspectos, son equivalentes a los paréntesis para agrupar y tienen la misma prioridad. Por ejemplo,
<polite> = please | kindly | oh mighty computer;
public <command> = [<polite>] don't crash;

Permite a un usuario decir " don't crash " y añadir opcionalmente una forma de cortesía como "oh mighty computer don't crash" o "kindly don't crash".

2.1.3 Ontologías de Dominio

2.1.3.1 Definición de Ontología

Las ontologías proveen una comprensión compartida y consensuada del conocimiento de un dominio que puede ser comunicada entre personas y sistemas heterogéneos. Las ontologías fueron desarrolladas en el área de Inteligencia Artificial (IA) para facilitar el intercambio y reutilización del conocimiento [59].

Una ontología es una jerarquía de conceptos con atributos y relaciones que tiene una terminología consensuada para definir redes semánticas de unidades de información interrelacionadas, proporcionando un vocabulario de clases y relaciones para describir un dominio. Por tanto, una ontología debe ser capaz de proveer un buen entendimiento del dominio que representa, esto incluye relaciones de términos y conceptos, sus definiciones o significados, sus relaciones con cada uno de ellos y las características del dominio [60].

Algunos problemas surgen cuando se trata con grandes cantidades de información semi-estructurada. Los actuales buscadores basados en palabras clave suelen devolver información irrelevante que usa una cierta palabra con un significado diferente del que se pretende en la búsqueda, y pierden información cuando no reconocen palabras diferentes, pero con el mismo significado que la buscada [61]. Pueden existir dos o más ontologías conectadas compartiendo información de dominios distintos, o se puede dar el caso que agentes de software requieran usar la información contenida en una ontología específica. Todo esto trae consigo la interoperabilidad entre diferentes sistemas de computación ayudando a un entendimiento compartido más amplio y reutilizable [62].

2.1.3.2 Componentes de las Ontologías

Los componentes de una ontología varían de acuerdo al dominio de interés y a las necesidades de los desarrolladores. Por lo general entre los componentes se encuentran los siguientes [63, 64]:

- **Clases:** son la base de la descripción del conocimiento en las ontologías ya que describen los conceptos (ideas básicas que se intentan formalizar) del dominio. Las clases usualmente se organizan en jerarquías a las que por lo general se les aplican mecanismos de herencia, es decir un concepto de nivel superior generaliza a otro de nivel inferior [65].
- **Relaciones:** Representan las interacciones entre los conceptos del dominio. Las ontologías por lo general contienen relaciones binarias; el primer argumento de la relación se conoce como el dominio y el segundo como el rango.
- **Funciones:** Son un tipo concreto de relación donde se identifica un elemento mediante el cálculo de una función que considera varios elementos de una ontología.
- **Instancias o individuos:** Representan objetos o elementos concretos de un concepto.
- **Taxonomía:** Conjunto de conceptos organizados jerárquicamente. Las taxonomías definen las relaciones entre los conceptos pero no los atributos de éstos.

- **Axiomas:** Se usan para modelar sentencias que son siempre ciertas. Los axiomas permiten, junto con la herencia de conceptos, inferir conocimiento que no esté indicado explícitamente en la taxonomía de conceptos. Los axiomas definidos en una ontología pueden ser estructurales o no estructurales: un axioma estructural establece condiciones relacionadas con la jerarquía de la ontología, conceptos y atributos definidos; un axioma no estructural establece relaciones entre atributos de un concepto y son específicos de un dominio. Los axiomas se utilizan también para verificar la consistencia de la ontología.
- **Propiedades** (Slots): Son las características o atributos que describen a los conceptos. Las especificaciones, rangos y restricciones sobre los valores de las propiedades se denominan *facets*. Para un concepto dado, las propiedades y las restricciones sobre éstos son heredadas por las subclases y las instancias de la clase.

2.1.3.3 Lenguajes de creación de Ontologías

A continuación se presentan algunos lenguajes utilizados en el desarrollo de ontologías [62]:

1. **XML (eXtensible Markup Language)**

XML es un meta-lenguaje derivado de SGML, que permite la definición de lenguajes de marcado adecuados para usos específicos y es un lenguaje basado en marcas tipo etiquetas. XML es una manera flexible de crear formatos de información comunes y compartir tanto los formatos como los datos entre sistemas de computación y es un estándar W3C (World Wide Web Consortium). La estructura de un documento XML se describe a través de DTDs (Definición de Tipo de Documento) o Schemas (Esquemas). Los DTDs poseen un lenguaje propio para su escritura, no admiten espacios de nombres (NameSpaces) y soportan tipos de datos muy limitados. Los Schemas utilizan sintaxis XML, admiten espacios de nombres y permiten definir tipos de datos simples, complejos y propios del usuario [66] .

2. **RDF (Resource Description Framework)**

RDF es una recomendación de la W3C para representar metadatos en la Web. Proporciona un medio para agregar semántica a un documento sin referirse a su estructura y es una infraestructura para la codificación, intercambio y reutilización de metadatos estructurados [67]. Debido a que RDF no define ningún vocabulario particular para la creación de los datos, por ello, se creó la especificación RDF Schema (RDFS) o lenguaje de descripción del vocabulario RDF.

El modelo de datos de RDF está formado por recursos (objetos) y pares de atributos/valores [67]. Un recurso representa cualquier entidad que pueda ser referenciada por un URI (Identificador Único de Recursos). Los atributos representan las propiedades de los recursos, y sus valores pueden ser entidades atómicas (por ejemplo: strings, enteros) u otros recursos.

RDFS carece de capacidades para describir la semántica de los conceptos y las relaciones más allá de aquella provista por los mecanismos de herencia. Además, solo provee las primitivas más básicas para el modelado de ontologías y no proporciona soporte para definir axiomas directamente. RDFS no es lo suficientemente expresivo para representar ontologías de gran complejidad.

3. XOL (Ontology Exchange Language)

Es un lenguaje para el intercambio de ontologías. Fue diseñado para ser utilizado como un lenguaje intermedio para permitir la transferencia de las ontologías entre diferentes sistemas de bases de datos, herramientas de desarrollo de ontologías o aplicaciones. Este puede ser utilizado para ontologías de cualquier dominio. La sintaxis de XOL está basada en XML y la semántica está basada en OKBC-Lite, que es una forma simplificada del modelo de conocimiento de OKBC (Open Knowledge Base Connectivity) [68].

4. OIL (Ontology Interface Layer)

Es un lenguaje de representación de ontologías basado en Web y capas de inferencia, que combina las primitivas de representación de conocimiento de los lenguajes basados en marcos con la semántica formal y los servicios de razonamiento proporcionados por la lógica descriptiva [67]. Incluye una semántica precisa para describir el significado de los términos. OIL unifica tres aspectos importantes proporcionados por diferentes comunidades como son: lógica descriptiva, sistemas basados en marcos y lenguajes Web:

- ***Lógica descriptiva*:** Un lenguaje formal para representar el conocimiento de un dominio de aplicación y razonar sobre el mismo.
- ***Sistemas basados en marcos*:** Sus primitivas de representación principales son las *clases* y sus *propiedades*. Las propiedades no tienen alcance global, sólo son aplicables en las clases en las que fueron definidas.
- ***Estándares Web (XML y RDF)*:** OIL tiene una sintaxis bien definida en XML basada en DTDs y Schemas. Además, es una extensión de RDF y RDFS. OIL está formado por 4 capas:
- ***Core OIL:*** coincide en gran parte con RDFS.
- ***Standard OIL:*** es un lenguaje que abarca las primitivas de representación necesarias para proporcionar un adecuado poder expresivo a la ontología. Especifica la semántica y hace viable la inferencia.
- ***Instance OIL:*** permite la definición de instancias en la ontología.
- ***Heavy OIL:*** reservada originalmente para posibles extensiones futuras de OIL. Puede incluir capacidades adicionales de representación y razonamiento.

Las principales limitaciones de OIL son [67]:

- Aunque OIL proporciona un mecanismo para heredar los valores de las superclases, estos valores no pueden ser sobrescritos en una especialización.
- Sólo un número fijo de propiedades algebraicas pueden ser expresadas sobre los atributos.
- No soporta dominios concretos (por ejemplo: números enteros).

5. DAML+OIL

Es un lenguaje de marcado semántico para recursos Web. Es un estándar propuesto por la W3C para la representación de ontologías y metadatos. DAML (DARPA Agent Markup Language) fue

transformado a DAML+OIL a través de la inclusión de algunas características de OIL al lenguaje. DAML consiste en un formalismo que permite a los agentes de software interactuar entre ellos [69].

DAML+OIL se construye sobre RDF y RDFS, pero proporciona primitivas de representación más ricas, comúnmente encontradas en la lógica descriptiva. DAML+OIL soporta tipos de datos complejos, a diferencia de OIL que sólo soporta el tipo de dato *string* [66], los tipos de datos utilizados provienen de XML Schema. El lenguaje tiene una semántica bien definida.

6. OWL (Web Ontology Language)

Es un lenguaje de marcado semántico desarrollado por la W3C para publicar y compartir ontologías sobre el World Wide Web. Es una extensión del vocabulario de RDF y se deriva de DAML+OIL. OWL está diseñado para ser utilizado por aplicaciones que necesitan procesar el contenido de la información en lugar de sólo presentarla a las personas [70].

OWL se puede formular en RDF, por lo que se suele considerar una extensión de éste. Además incluye toda la capacidad expresiva de RDF y la extiende con la posibilidad de utilizar expresiones lógicas [71]. Proporciona tres sub-lenguajes diseñados para ser utilizados por comunidades específicas de desarrolladores y usuarios. La característica que define a cada lenguaje es su expresividad [61]:

- ✓ ***OWL Lite:*** Es el sub-lenguaje con sintaxis más simple, su intención es ser utilizado en situaciones donde se requiera una jerarquía de clases y restricciones simples. Tiene restricciones simples (cardinalidad sólo 0 o 1), facilita compatibilidad con otros modelos/paradigmas, facilita desarrollo de herramientas de autor. Tiene un razonamiento eficiente, facilita desarrollo de herramientas, compatibilidad con otros modelos.
- ✓ ***OWL DL*** *(Description Logics):* Es mucho más expresivo que OWL Lite y está basado en lógica descriptiva. Proporciona la máxima expresividad posible sin perder la completitud computacional (todas las conclusiones pueden ser deducidas) y la posibilidad (todos los cálculos se realizan en un tiempo finito). Es favorable para el razonamiento automático, para la clasificación de jerarquías y para detectar las inconsistencias en las ontologías. No puede haber restricciones de cardinalidad (locales ni globales) en propiedades transitivas, ni sus inversas, ni sus superpropiedades.
- ✓ ***OWL Full:*** Es el sub-lenguaje más expresivo, su intención es ser utilizado en situaciones donde una alta expresividad es más importante que la capacidad de garantizar la completitud computacional y la posibilidad. Realiza unión de sintaxis OWL y RDF (sin restricciones).

2.1.3.4 Herramientas y API's de programación de ontologías

1. Protégé

Protégé es un software libre de código abierto implementado en Java, desarrollado en la Universidad de Stanford, que permite la construcción de ontologías de dominio. Es capaz de operar como una plataforma para acceder a otros sistemas basados en conocimiento o aplicaciones integradas, o como una librería que puede ser usada por otras aplicaciones para acceder y visualizar bases de conocimiento. La herramienta ofrece una interfaz gráfica que permite al desarrollador de ontologías

enfocarse en la modelación conceptual sin que requiera de conocimientos de la sintaxis de los lenguajes de salida. Protégé ha sido utilizado como el ambiente de desarrollo primario para muchas ontologías, y se ha convertido en la herramienta más utilizada en el mundo para trabajar con OWL.

Protégé presenta una gran capacidad de extensión, debido al soporte de conectores (plug-ins), que son aditivos que se adquieren de manera individual y se acoplan al entorno de trabajo de Protégé para añadirle funcionalidad. Existen varios conectores disponibles para importar ontologías en diferentes formatos, incluyendo DAG-EDIT, XML, RDF y OWL.

Protégé puede ser conectado directamente a programas externos con la finalidad de utilizar sus ontologías en aplicaciones inteligentes, tales como servicios de razonamiento y clasificación. Los desarrolladores de sistemas pueden utilizar la Interfaz de Programa de Aplicación (API) Java de Protégé (Protégé OWL API) para acceder y manipular las ontologías [62]. En este sentido, la API de Protégé-OWL es una biblioteca Java de código abierto para el Lenguaje de Ontologías Web y RDF (S), proporciona clases y métodos para cargar y guardar archivos OWL, para consultar y manipular modelos de datos OWL, y llevar a cabo el razonamiento [72].

2. **OWL API**

OWL API [73] es una interfaz Java y la implementación para el Lenguaje de Ontologías Web del W3C (OWL. OWL API también ofrece enlaces específicos para razonadores OWL DL: Fact++ y Pellet. Incluye los siguientes componentes:

- Parser RDF/XML parser
- Parser OWL /XML
- Analizador de sintaxis funcional OWL
- Soporte para la integración con los razonadores como Pellet y FACT++.

3. **Jena**

Ver apartado Jena, en la sección “Razonadores de Programación Lógica”.

2.1.3.5 Formalismos y razonamiento con ontologías

Para que los programas software sean capaces de realizar procesos de inferencia sobre la información almacenada en las ontologías, éstas deben de fundamentarse en algún formalismo lógico. La utilización de estos formalismos lógicos, ofrecen la posibilidad de realizar procesos de razonamiento e inferencia que conducen a la adquisición de nuevo conocimiento a partir del ya existente. Así, se puede decir que una lógica define un lenguaje formal para expresar conocimiento acerca de algún dominio y las proposiciones a ser derivadas de este conocimiento, es decir, fija el conjunto de primitivas que pueden emplearse para modelar el contexto, esto es, la *sintaxis*, y establece la *semántica* formal de cada proposición en el lenguaje correspondiente (valor de verdad de cada sentencia respecto a cada mundo posible) y está equipada con un ‘*cálculo*’, es decir, un procedimiento formal (computable) para derivar nuevos hechos (descritos en el lenguaje correspondiente) a partir de un conjunto de proposiciones dado en el lenguaje. Los tres formalismos lógicos más relevantes son: *lógica de primer orden, lógica descriptiva* y *programación lógica* [74].

1. Lógica de Primer Orden (FOL)

En [75] se indica que la Lógica de Primer Orden puede utilizarse para representar los modelos que pertenecen a un dominio particular a través de sentencias y realizar razonamiento sobre este a través de dichas sentencias para producir nuevos modelos del domino.

La LPO se vale de la representación de los objetos, relaciones entre dichos objetos y la teoría del cálculo de predicados para realizar inferencias sobre el domino. Los elementos básicos de la LPO son símbolos que representan los objetos y relaciones. Para representar el domino se define el vocabulario de los predicados, constructores y objetos y se construyen sentencias compuestas por términos y constructores [76]. Entre las ventajas de la LPO se encuentran las siguientes:

- Estructura el mundo en objetos y relaciones, lo que facilita el razonamiento.
- Ofrece libertad para describir el mundo de la manera que el diseñador considere apropiada.
- Permite expresar sentencias sobre todos los objetos del universo.

2. Lógicas Descriptivas (DL)

La Lógica Descriptiva es un formalismo de representación del conocimiento caracterizado por describir los conceptos relevantes de un domino particular y utilizar estos para especificar las propiedades de los objetos e instancias del dominio [77].

Las Lógicas Descriptivas se construyen a partir de un conjunto de clases (conceptos), relaciones (roles o propiedades), e instancias (individuos). Conceptos atómicos y roles se utilizan para representar un concepto primordial en el dominio o una relación binaria entre ellos. Términos más complejos se construyen utilizando operadores, como intersección, unión, complemento, y restricciones de valor que combinan términos atómicos y complejos en conjunto para definir nuevas clases.

Básicamente un Sistema Basado en Lógica Descriptiva (SBDL) se caracteriza por tener una base de conocimiento (KB) compuesta por dos componentes. El primero, la *T-Box* (Terminological knowledge box) el cual indica la terminología del dominio y se refiere a declaraciones con respecto a ésta terminología, en él se describe la jerarquía de conceptos y las relaciones entre ellos [78]. El segundo, la *A-Box* (Assertional knowledge box), que contiene las aserciones acerca de las instancias particulares del domino con base en el vocabulario expresado en la terminología.

Uno de los objetivos de la Ingeniería del Conocimiento es apoyarse en los motores de razonamiento para explotar los datos anotados semánticamente y por lo tanto deducir nuevo conocimiento específico a un dominio representado por la ontología. Los razonadores DL deben proporcionar los siguientes servicios de inferencia [79-81]:

- **Validación de la consistencia de una ontología.** El razonador debe ser capaz de asegurar que una ontología no contiene hechos contradictorios.
- **Validación del cumplimiento de los conceptos de la ontología.** El razonador determina si es posible que una clase tenga instancias. En el caso de que un concepto no sea de satisfacción, la ontología será inconsistente.

- **Clasificación de la ontología.** El razonador computa, a partir de los axiomas declarados en la *T-Box* de la ontología, las relaciones de subclase entre todos los conceptos declarados explícitamente para construir la jerarquía de clases.
- **Precisiones sobre los conceptos de la jerarquía**. Un razonador DL puede inferir cuáles son las clases a las que directamente pertenece. Cuando utiliza la jerarquía inferida mediante el servicio de clasificación previamente descrito, es posible obtener todas las clases a las que indirectamente pertenece una instancia dentro de la ontología.

En conclusión, las tareas de razonamiento tienen como objetivo la explotación significativa de las bases de conocimiento a través de la validación o la deducción de nuevos conocimientos [78]. Entre estos motores o razonadores se tienen:

A. Pellet

Es un razonador *open source* para OWL-DL completo con muy buen rendimiento [79], y se encuentra construido en Java. Pellet soporta toda la expresividad de la lógica descriptiva, implementa estrategias de *T-Box*, soporte nominal (clases definidas por extensión), absorción, ramificación semántica. Su razonamiento individual, de tipos de datos (DataType), y la optimización de las respuestas a las consultas de *A-Box*, hacen que sea adecuado para aplicaciones web semánticas de sonido [17, 42, 46].

Pellet soporta la gama estándar de las consultas derivativas y varios servicios de gestión de razonamiento a través de su propia API, mediante el suministro de enlaces y el apoyo de toolkits comunes (por ejemplo, Jena [82], WonderWeb de OWL API [83] y DIG [84]). También se ha ido más allá tanto en el conjunto estándar de servicios de inferencia (consistencia, satisfacibilidad, clasificación, y la realización) y las recomendaciones sugeridas por la W3C (la coherencia, la vinculación, y respuesta a consultas conjuntivas) para introducir varios servicios estándar, que se cree que son casi indispensables para el uso práctico.

La Figura 4 muestra los principales componentes de pellet. El núcleo del sistema es el razonador *Tableaux* que chequea la consistencia de una base de conocimientos. El razonador está acoplado con un tipo de datos oracle que puede comprobar la consistencia de las conjunciones (built-in o derivados) de los tipos de datos de XML esquema simple. Las ontologías OWL se cargan en el razonador después de la validación y reparación de ontología. Durante la fase de carga, los axiomas sobre las clases se almacenan en la *T-Box* y las afirmaciones acerca de las instancias en la *A-Box*. Los axiomas *T-Box* pasan por el pre-procesamiento estándar de razonadores DL.

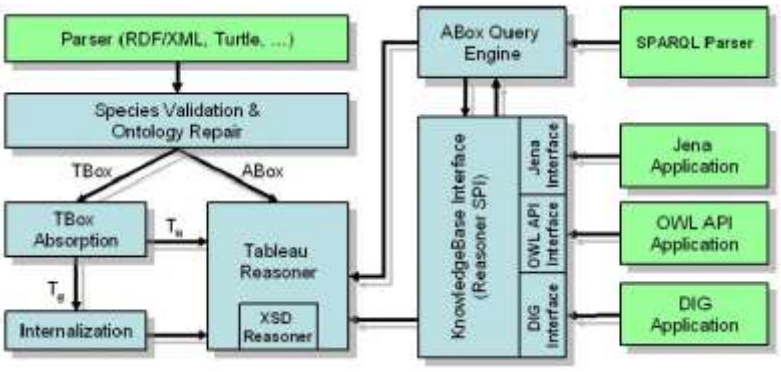

Figura 4. Principales componentes del razonador Pellet

El sistema proporciona una fina capa de acceso a través de la interfaz de programación de servicio (SPI), que ofrece funciones convenientes para acceder a los servicios de razonamiento [79].

Este razonador implementa las mejores técnicas de optimización, lo que hace que su desempeño sea bueno, en especial cuando debe evaluar ontologías con mayor complejidad y expresividad; sin embargo no es tan eficiente como RacerPro o FACT ++ en clasificaciones [79].

B. Fact++

Es un razonador DL que soporta la lógica *SHOIQ(D)* y ha sido desarrollado bajo el proyecto europeo WonderWeb" [81, 85]. Está implementado en C++ y corresponde a una nueva versión del razonador DL FaCT, que fue originalmente implementado en Lisp [35]. FaCT++ es un buen razonador para la *T-Box* de una ontología. Sin embargo, hay dos desventajas importantes de FaCT++, por un lado, no tiene soporte para otros tipos de dato que no sean *string* o *integer*, como sí ocurre, por ejemplo, con Pellet, y por otro lado, tampoco tiene soporte para el razonamiento con la *A-Box* de la ontología.

Entre las ventajas que ofrece FaCT++ se tienen: posee licencia GPL, que todavía tiene mantenimiento y sigue en desarrollo (actualmente se está investigando cómo aumentar la capacidad de razonamiento del motor de inferencia y tiene soporte de OWL2) [85] y trabaja de forma eficiente con la *T-Box* de ontologías de tamaño grande y mediano.

C. Racer Pro

Por sus siglas, Renamed Abox and Concept Expression Reasoner (RACER) es un razonador DL para la lógica descriptiva *SHI*, desarrollado en LIPS. Inicialmente fue creado por la Universidad de Hamburgo, pero luego se convirtió en software propietario y su nombre comercial es RacerPro [86, 87]. Soporta OWL DL excepto para los nominales (clases definidas por una enumeración de sus miembros, que implementa como definiciones parciales) y para tipos de datos no estándar; maneja largas *A-Boxes* en combinación con largas y expresivas *T-Boxes* y provee servicios de inferencia sofisticados [65, 79].

3. Programación Lógica

En ésta, se trabaja de una forma descriptiva, estableciendo relaciones entre entidades e indicando no el cómo se deben de hacer las cosas, sino el qué hacer. La programación lógica se basa en un subconjunto de LPO denominada '*Lógica de Horn*' [88]. Sin embargo, la semántica de la programación lógica es diferente de la LPO. En este caso, la semántica está basada en los modelos mínimos de Herbrand.

Por analogía con la formulación implicativa y deductiva, la notación utilizada para este tipo de cláusulas es una lista de literales negados como antecedente y el literal afirmado como consecuente separados por el signo de implicación: $A(x),D(x),E(x) \rightarrow F(x, y)$. De esta forma, un programa lógico consiste en reglas de la forma 'si A entonces B'.

El uso de motores de reglas, conocido también como motores de inferencia, es un paradigma general basado en datos, y por tanto, es declarativo para deducir nuevas conclusiones a partir de un conjunto de datos y reglas de inferencia. Una regla de inferencia generalmente se expresa a través de una cláusula "If" y una cláusula "Then". Hay dos grandes tipos de sistemas basados en reglas: los sistemas de encadenamiento hacia adelante y los de encadenamiento hacia atrás. El encadenamiento hacia adelante empieza con los primeros hechos y sigue utilizando las reglas para sacar nuevas conclusiones, dados los hechos. En un sistema de encadenamiento hacia atrás, se comienza con algunas hipótesis (o metas) que se están tratando de probar, y sigue buscando reglas que le permiten concluir la hipótesis y establece nuevos sub-objetivos [89].

A. Lenguajes de reglas

La definición de reglas juega un papel importante en los procesos de inferencia de ontologías: de la buena definición de las reglas depende el éxito para la generación de nuevo conocimiento [90]. A través de las reglas se expresa el comportamiento de las instancias dentro de un dominio. Se detalla a continuación algunos de los lenguajes de reglas:

- ✓ **SWRL.**

SWRL, por sus siglas en inglés *Semantic Web Rule Language*, se basa en la combinación de OWL DL- OWL Lite, OWL DL y RuleML [91]. Para extender el OWL utiliza las reglas *Horn-Like* (de tipo si.. entonces.. sino…) expresadas en términos de OWL (clase, literales, instancias, propiedades, etc.). Un editor de este lenguaje es Protegé [92, 93]. SWRL ofrece una sintaxis abstracta que incluye axiomas y hechos, por esto, las reglas son parejas formadas por un antecedente (cuerpo) y un consecuente (encabezado). Cada consecuente puede estar formado por cero o más átomos, y éstos pueden ser instancias, literales, variables de instancias o variables de datos [90].

- ✓ **DL Safe Rules**

Puede considerarse como un subconjunto de SWRL (Semantic Web Rule Language), en la que las variables pueden solo asumir valores concretos. Surgieron como una extensión de OWL–DL, con el objetivo de asegurar que un razonador no llegue a un estado de no decisión [94]. Las reglas *DL Safe* utilizan reglas *Horn* con predicados binarios y unarios [95]. Estas reglas tienen una expresividad limitada; por ejemplo, no permiten la negación o disyunción en el cuerpo de la regla [96]. Son soportadas por el razonador KAON2.

- ✓ **RuleML**

Por sus siglas en inglés Rule Markup Language, es un lenguaje para representar reglas de negocio en la *web* y fue definido por Rule Markup Initiative. Cubre un amplio rango de reglas que incluye reglas de derivación, reglas de transformación y reglas de reacción [97]. La idea de RuleML es permitir realizar tareas de deducción e inferencia sobre la web, mapeos entre ontologías y comportamientos dinámicos como servicios y agentes. La sintaxis de RuleML está basada en las

definiciones de XML esquema y su núcleo o kernel es Datalog, que constituye un sublenguaje de la lógica de *Horn*.

- ✓ **Jess**

Por sus siglas en inglés, Java ExpertSystem Shell es un lenguaje de programación basado en CLIPS, creado por Ernest Friedman- Hill en Sandia National Laboratories. Su funcionamiento se basa en reglas, utiliza una memoria de trabajo que es similar a una base de datos y usa una versión mejorada del algoritmo Rete (Algoritmo de Redundancia Temporal), que es utilizado para realizar inferencia empleando el método de encadenamiento hacia delante (forwards chaining); con él se aplica al conocimiento base (base de hechos) otro conocimiento (las reglas) para obtener nuevos resultados. Jess puede utilizarse como un motor de reglas o como lenguaje de desarrollo para sistemas expertos y se puede integrar a Protegé a través del *plugin* JessTab [98, 99].

B. Razonadores de Programación Lógica

A continuación, se presenta una lista con los razonadores más habituales para la programación lógica.

- ✓ **KAON2**

Es un razonador con un sistema de razonamiento híbrido que soporta SHIQ(D), un subconjunto de OWL-DL, Datalog y DL-safe [81, 100]; fue desarrollado por la Universidad de Manchester, la Universidad de Karlsruhe y el FZI. [101] Dentro de KAON2, las reglas DL-Safe pueden ser agregadas a la salida de un DataLog Disyuntivo y no requieren preprocesamiento adicional, lo que mejora el rendimiento [101]. Su razonamiento es implementado a través de algoritmos Novel, que reducen una base de conocimiento SHIQ(D) a un programa de registro de datos disyuntivos [6][100]. KAON no maneja nominales ni números grandes en sentencias de cardinalidad [100].

- ✓ **FLORA-2**

Es un sistema basado en conocimiento y un entorno completo para el desarrollo de aplicaciones que hacen uso de conocimiento. FLORA-2 extiende el cálculo de predicados clásico con el concepto de objeto, clase y tipo, que han sido adaptados de la programación orientada a objetos [101].

- ✓ **Jena**

Es un framework de Java *open source* para construir aplicaciones de *web* semántica sobre modelos RDF [24]. Jena proporciona a los desarrolladores un API para leer y escribir en formatos XML/RDF, N3 y N-Triples (formatos para grafos abreviados) [102]. Adicionalmente, permite el tratamiento del lenguaje OWL, al utilizar el modelo semántico de RDF, y proporciona conexión de forma externa a través del interfaz DIG hacia razonadores DL, como Pellet o FaCT++ [81]. Jena posee la capacidad para conectarse con motores de inferencia externos tipo Plug-in a través de una interface DIG; sin embargo, tiene su propio subsistema de inferencia que consiste en un motor

híbrido con encadenamiento *hacia-adelante* y *hacia-atrás* utilizando el algoritmo RETE (algoritmo de reconocimiento de Patrones). Los razonadores que ofrece son [81]:

- Razonador transitivo, que implementa propiedades transitivas y simétricas de rdfs: subPropertyOf and rdfs: subClassOf [103]
- Razonador para RDF(S)
- Razonador para OWL
- Razonador para DAML (Darpa Agent Markage Language) [104]

Motor de reglas multipropósito, que puede ser utilizado para el procesamiento de RDF, deducción de nuevo conocimiento y la transformación de grafos RDF.

2.1.3.6 Consulta a Ontologías

Los lenguajes de ontologías y sus lenguajes de consulta correspondientes juegan un papel clave para la representación y el procesamiento de la información sobre el mundo real. Los lenguajes de consulta de ontologías se han desarrollado para consultar la información definida por estos lenguajes de ontologías y sistemas de razonamiento [105].
La capacidad de recuperar los datos modelados por los lenguajes anteriormente mencionados es un aspecto muy importante, ya que la respuesta a consultas es un proceso complejo debido a algunas particularidades de la Web [106], además, la consulta debe tener en cuenta el significado que se define por los metadatos y tiene que entender y procesar adecuadamente la misma. Para lograr esto hay varios lenguajes de consulta, y se presentan a continuación.

1. **RDQL**

RDQL es un lenguaje de consulta para RDF basado en SquishQL en el framework Jena. El desarrollo de RDQL es para proporcionar un modelo de consulta orientado a datos. Esto significa que RDQL sólo recupera la información almacenada en el modelo que contiene un conjunto de declaraciones N-Triple [107] y no proporciona mecanismos de razonamiento. Aunque RDQL es relativamente simple en la sintaxis, es eficaz para la mayoría de las consultas de la ontología y su sintaxis simple lo hace muy flexible [105].

2. **SPARQL**

SPARQL [57] está estandarizado a través de una recomendación W3C y es el lenguaje de consulta RDF más popular. Las consultas se componen de dos cláusulas, la primera, especifica el tipo de la consulta y la segunda, la cláusula WHERE, consiste en definir los patrones a través de tres variables para identificar los datos de destino. Las consultas pueden incluir conjunciones, disyunciones, u opcionalidad [78].

Las especificaciones de SPARQL no existen aisladamente, ya que hay varias herramientas y APIs que ya proporcionan la funcionalidad de SPARQL, y la mayoría de ellas están al día con las últimas especificaciones. Una breve lista incluye:

- ARQ, un procesador de SPARQL para Jena

- Rasqal, la biblioteca de consultas RDF incluidos en el framework de DaveBeckett Redland
- twinql, un procesador de SPARQL para Lisp escrito por Richard Newman
- Pellet, cuenta con el apoyo parcial de consulta SPARQL
- KAON2, otro razonador OWL DL que cuenta con el apoyo parcial de SPARQL [108].

3. nRQL

nRQL (nuevo lenguaje de consulta RacerPro, una extensión de RQL) [109] es un lenguaje de consulta ampliada de RACER. nRQL se construyó sobre la base de la teoría de la descripción del modelo lógico [110]. Por lo tanto, nRQL es:

- Un lenguaje de consulta expresivo para ABox
- Un lenguaje de consulta RDF (S)
- Un lenguaje de consulta OWL.

Las características del lenguaje nRQL se pueden resumir de la siguiente manera [111]:

- nRQL tiene una sintaxis y semántica bien definidas.
- nRQL ofrece una variedad de átomos de consulta, las más importantes son: átomos de consulta de concepto, átomos de consulta de roles, átomos de consulta de restricciones.
- Apoyo especial para consulta de la parte del dominio concreto de una *A-Box*
- nRQL es también un poderoso lenguaje de consulta OWL y RDF (S).

4. SQWRL (Semantic Query-enhanced Web Rule Language)

SQWRL, está basado en el lenguaje de reglas SWRL y utiliza la base semántica sólida SWRL como sus fundamentos formales. El lenguaje resultante proporciona un pequeño pero poderoso conjunto de operadores que permite a los usuarios construir consultas en ontologías OWL. SQWRL también contiene un conjunto de operadores que se pueden utilizar para realizar operaciones de conclusión para permitir formas limitadas de la negación como fracaso, cálculo y agregación.

SQWRL toma un antecedente de una regla SWRL estándar y eficazmente lo trata como una especificación del patrón para una consulta. Éste reemplaza el consecuente de una regla con una especificación de recuperación y utiliza la habilidad SWRL incorporada como un punto de extensión [6].

Una implementación de SQWRL se ha desarrollado en el SWRL Tab Plugin [112] en Protégé-OWL. La implementación proporciona una interfaz gráfica para editar y ejecutar consultas SQWRL y también proporciona una interfaz similar a JDBC Java para ejecutar consultas SQWRL en las aplicaciones Java.

2.2 TRABAJOS RELACIONADOS

A continuación, se resumen los trabajos relacionados más relevantes en el contexto del presente proyecto.

2.2.1 "Development Issues for Speech-Enabled Mobile Applications"

[10] presenta un enfoque integral que comprende una clasificación profunda de los sistemas de reconocimiento de voz para aplicaciones móviles, y un framework para reconocimiento de voz distribuida y móvil. Se presenta la arquitectura del mismo, en la que se separa el desarrollo de la lógica del negocio del cliente de la capa de reconocimiento del habla. Los autores resaltan, que este framework ofrece características adicionales comparadas con componentes de interfaz de usuario gráficos comunes, ya que permite el envío de eventos si se operan mediante la voz y permite a un desarrollador asociar una gramática (vocabulario con restricciones) o permitir el dictado (vocabulario sin restricciones). Finalmente, para comprobar su funcionamiento, se presenta el desarrollo de una aplicación empresarial basada en el framework y su componente de interfaz de usuario multi-modal, explicando cómo y dónde implementar los componentes del framework. El marco se implementó sobre dispositivos Microsoft Windows Mobile Pocket PC y Smartphone Edition.

Este trabajo aporta un framework que acelera el proceso de desarrollo, con componentes multi-modales, que pueden ser fácilmente utilizados por un diseñador de interfaz gráfica de usuario; adicionalmente, define capas de abstracción que soportan la integración de varios motores de reconocimiento de voz que pueden ser locales (dispositivo móvil) o remotos (servidor) dependiendo de las necesidades del usuario. Sin embargo este framework está dirigido exclusivamente a permitir la creación de interfaces de voz de usuario, enfocándose en la utilización de un motor de reconocimiento de voz, que puede ser distribuido o móvil, sin tener en cuenta el contexto de la aplicación como tal.

2.2.2 "Ontology driven voice-based interaction in mobile environment"

[16] presenta un prototipo de interfaz de voz que permite a los usuarios interactuar con el sistema móvil de gestión del conocimiento, que se ha desarrollado en la Universidad Técnica Checa en Praga. El dominio primario de aplicación del sistema es la facilidad y la gestión de construcción de edificaciones. Las descripciones semánticas junto con la ontología de dominio se almacenan en el formato OWL.

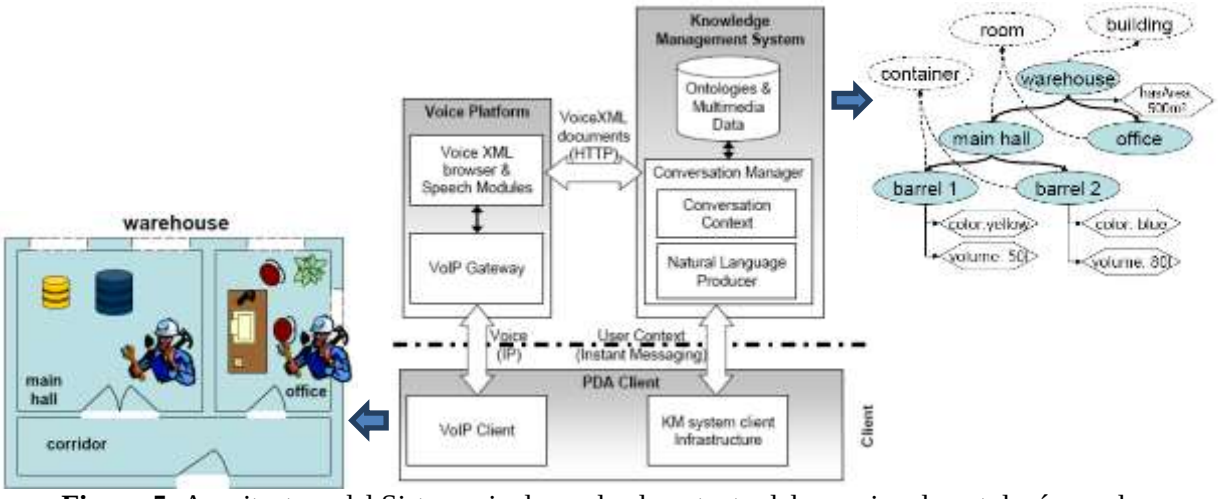

Figura 5. Arquitectura del Sistema, incluyendo el contexto del usuario y la ontología que lo representa. Fuente [16].

La arquitectura que se presenta utiliza un servidor VoiceXML, que es la plataforma de reconocimiento de voz y síntesis. Está configurado para solicitar documentos VoiceXML del sistema, que implementa la lógica de la interacción, la administración del estado de conversación y la recuperación de datos de ontología.

Este trabajo aporta una interfaz de voz de usuario que utiliza información contextual para hacer el proceso de comprensión eficiente, ya que en un contexto determinado, sólo una parte relevante del lenguaje puede ser utilizado; no obstante, por haber sido desarrollado en el año 2005, no considera aspectos relacionados con la plataforma Android.

2.2.3 " TVIS: Tactical Voice Interaction Services for Dismounted Urban Operations"

TVIS es presentado en términos de una arquitectura de sistema operacional, la cual incluye un vocabulario de gramática y la correlación entre la ontología, la gramática y la operación a realizar. Estos componentes se utilizan para ilustrar el acceso de voz, es decir la capacidad de un soldado en campo para acceder a información táctica con los servicios de reconocimiento de voz [17].

La arquitectura del sistema es cliente-servidor, una consulta de voz se origina desde un dispositivo móvil, limitando el vocabulario por medio de una gramática predefinida, la cual es una instanciación de la ontología de voz que describe los principales componentes del dominio táctico.

Este trabajo aporta un sistema que potencia a los soldados desmontados con la capacidad de acceder de forma autónoma y recuperar información ISR usando un servicio personalizado de respuesta de voz interactiva (IVR), refiriéndose como un servicio de interacción táctica de voz (TVIS), empleando una gramática predefinida representada por medio de ontologías de domino para restringir el lenguaje de entrada de los sistemas de reconocimiento de voz; no obstante, no considera aspectos relacionados con la plataforma Android.

2.2.4 "Ontology-based Context Inference and Query for Mobile Devices"

[113] describe un razonamiento de información de contexto y un componente de interfaz de consulta como parte del framework Service Context Manager (SCM) en ambientes de comunicación móvil, que soporta la consulta semántica y maneja la información incompleta del contexto a través de un mecanismo basado en reglas SWRL.

En la arquitectura propuesta, se mapea la información de contexto a una estructura formal basada en ontologías, que es almacenada en una Base de Datos OWL, la cual representa el vocabulario de dominio en términos de clases y propiedades.

[113] aporta un enfoque de inferencia de contexto, que consiste de un módulo de razonamiento que hace referencia a una base de reglas para deducir hechos de la ontología, y posteriormente se realiza la consulta para obtener la información deseada. Cabe señalar que las reglas están representadas de forma independiente del motor de inferencia y que las consultas han sido diseñadas a través de la librería integrada SQWRL. Sin embargo, a pesar que ésta es una de las iniciativas para realizar el tratamiento de ontologías en dispositivos móviles, no se exploran otras alternativas.

2.2.5 "Exploiting Context Information in Spoken Dialogue Interaction with Mobile Devices"

[114] describe la implementación de dos aplicaciones móviles concretas, un sistema de gestión de calendario, y la otra relacionada con colecciones de fotos, para ejemplificar el uso de una interfaz de voz como medio natural para controlar y comunicarse con dispositivos móviles. Se presentan diferentes alternativas en las cuales la información contextual puede utilizarse para mejorar la efectividad de diálogo hombre-dispositivo. Se utilizaron dispositivos Nokia Linux N800 con pantallas táctiles .

Este trabajo no utiliza Ontologías de Dominio para la creación de sus aplicaciones, pero aporta una serie de lineamientos con una visión muy propia de sus autores, acerca de la interacción intuitiva que permiten a los seres humanos comunicarse con sus teléfonos móviles, para acceder a los servicios, mencionando varias maneras en que la información contextual puede ser aprovechada para mejorar la efectividad del diálogo hombre-ordenador.

2.2.6 "Performing Intelligent Mobile Searches in the Cloud using Semantic Technologies"

En [115] se presenta un nuevo enfoque para realizar búsquedas móviles inteligentes combinando ontologías, computación en la nube y acceso móvil. Un prototipo funcional de un sistema de búsqueda restaurante móvil es presentado para la demostración del concepto. El prototipo construido es inteligente en la comprensión y asistencia en la elección de un restaurante que ofrece la máxima satisfacción al usuario bajo las restricciones dadas.

El sistema está diseñado para ser una aplicación móvil tipo cliente/servidor, donde el servidor se ejecuta en la nube, y filtra los resultados a través de la combinación de los comentarios de los clientes y las categorías predefinidas, para restringir la búsqueda si existen muchos restaurantes que cumplan los criterios o ampliarla si los resultados son muy pocos. El prototipo emplea Protégé 3.4 para la creación de la ontología, comprobar su consistencia y ejecutar consultas de ejemplo, Pellet 1,5 como motor de razonamiento, Java API para el resto de la computación en la nube, y el API de Android para el acceso móvil.

El anterior trabajo relaciona sistemas que emplean las ontologías en entornos móviles, y específicamente emplea la plataforma de desarrollo Android, sin embargo no se enmarca dentro del uso de sistemas de reconocimiento de voz.

2.2.7 "A Design and Implementation of Search Engine for Mobile Devices Based Chinese Semantics and Reasoning"

[116] propone un framework de infraestructura y describe el diseño de un prototipo, que permite a las personas consultar y organizar los recursos web basados en la semántica china para realizar razonamiento. Se emplea una ontología para abordar el razonamiento del conocimiento que se utiliza en los motores de búsqueda con el fin de obtener los resultados apropiados que pueden satisfacer las expectativas e intenciones de los usuarios.

La implementación del sistema presentado se compone de un motor de búsqueda y una aplicación cliente móvil, la cual emplea como herramientas para su construcción: JDK1.6, Apache Tomcat 6.0,

Lucene para construir motores de búsqueda, protégé para construir una ontología de dominio de acuerdo con el "Método de Clasificación de la Biblioteca Nacional de China", Sun WTK2.2 para desarrollar aplicaciones clientes móviles y Stock API para comunicar el terminal móvil con los servicios.

Este trabajo se centra en el uso de las tecnologías de ontologías para construir motores de búsqueda en dispositivos móviles, con el fin de proporcionar a los equipos, entendimiento de las consultas de los usuarios. Sin embargo, esta investigación no emplea tecnologías de reconocimiento de voz.

CAPÍTULO 3

EVALUACION DE LAS CAPACIDADES DE CONFIGURACION DE LOS MOTORES ASR Y LOS COMPONENTES DE RAZONAMIENTO ONTOLOGICO.

3.1 INTRODUCCION.

El presente capítulo presenta la evaluación de los módulos necesarios para el procesamiento de la gramática de contexto de una aplicación móvil basada en voz para clientes Android usando ontologías de dominio, los cuales comprenden los motores de reconocimiento de voz y los componentes de razonamiento ontológico; para la evaluación de éste último es necesario elegir una ontología de dominio de prueba y por supuesto realizar su evaluación, para determinar la consistencia y calidad en su construcción. El propósito de la evaluación del rendimiento de algunos motores ASR de código abierto es seleccionar los que presenten mejores capacidades de configuración y reconocimiento de voz sobre la plataforma Android, mientras que el de los componentes del razonamiento ontológico es seleccionar un mecanismo que procese la gramática de contexto de la aplicación.

3.2 ELECCIÓN DE LA ONTOLOGIA DE DOMINIO

3.2.1 Generalidades

La utilización de las ontologías en aplicaciones de negocios, en la Web semántica, para la administración, integración y reutilización de conocimiento, entre otras, ha ido incrementándose cada vez más y su uso se ha extendido a nuevas áreas. Así mismo, la reutilización de dichas ontologías en un dominio de interés, es bastante común entre los Desarrolladores de Ontologías (DO) debido al costo y complejidad que demanda la construcción de ontologías desde cero. Sin embargo, un inconveniente que por lo general se debe enfrentar, es que para un mismo dominio las conceptualizaciones pueden presentar algunas diferencias y por lo tanto las ontologías varían entre sí. Ante esta diversidad los DO deben elegir cuál ontología reutilizar y para ello necesitan evaluar las que se encuentran disponibles y seleccionar aquella que mejor se adapte a sus requerimientos [117].

Son muchos y variados los métodos de evaluación de ontologías [118], los cuales, por lo general, se basan en la comparación con ontologías de referencia. No obstante, cuando la ontología que se está desarrollando es la única descripción que se conoce del dominio, es decir, no existen precedentes, los desarrolladores se ven en la necesidad de utilizar de manera parcial los métodos de evaluación disponibles, lo cual no garantiza que la evaluación sea lo suficientemente completa y confiable, ya que como se dijo anteriormente, se tiene en cuenta una ontología de referencia.

Para el presente trabajo de grado se tomó como referencia una ontología de Arte por estar dentro del contexto del habla hispana, además tras una búsqueda exahustiva de Ontologias que fueran consistentes y livianas para propósitos de las pruebas a realizar, era la que cumplia con estos principales objetivos, adicionalmente, la temática del Arte, está muy relacionada con el aspecto colonial y cultural de la Ciudad de Popayán, donde se resalta el legado español por medio de la obras arquitectónicas contruidas del centro histórico, y las piezas artisiticas y religiosas que se exponen en los diferentes museos e iglesias de la ciudad, este es entonces, un buen contexto para ser elegido. Para

este caso, por ser la única descripción conocida en el dominio seleccionado razón por la cual se realizó su evaluación utilizando el esquema propuesto en [119], método construido para facilitar la evaluación de ontologías como descripciones únicas de un dominio, sin necesidad de contrastar con ontologías de referencia, y garantizando eficacia en cada proceso evaluativo. En este sentido, se tomó como ontología base, la ontología "Arte.owl", la cual tiene la siguiente jerarquía:

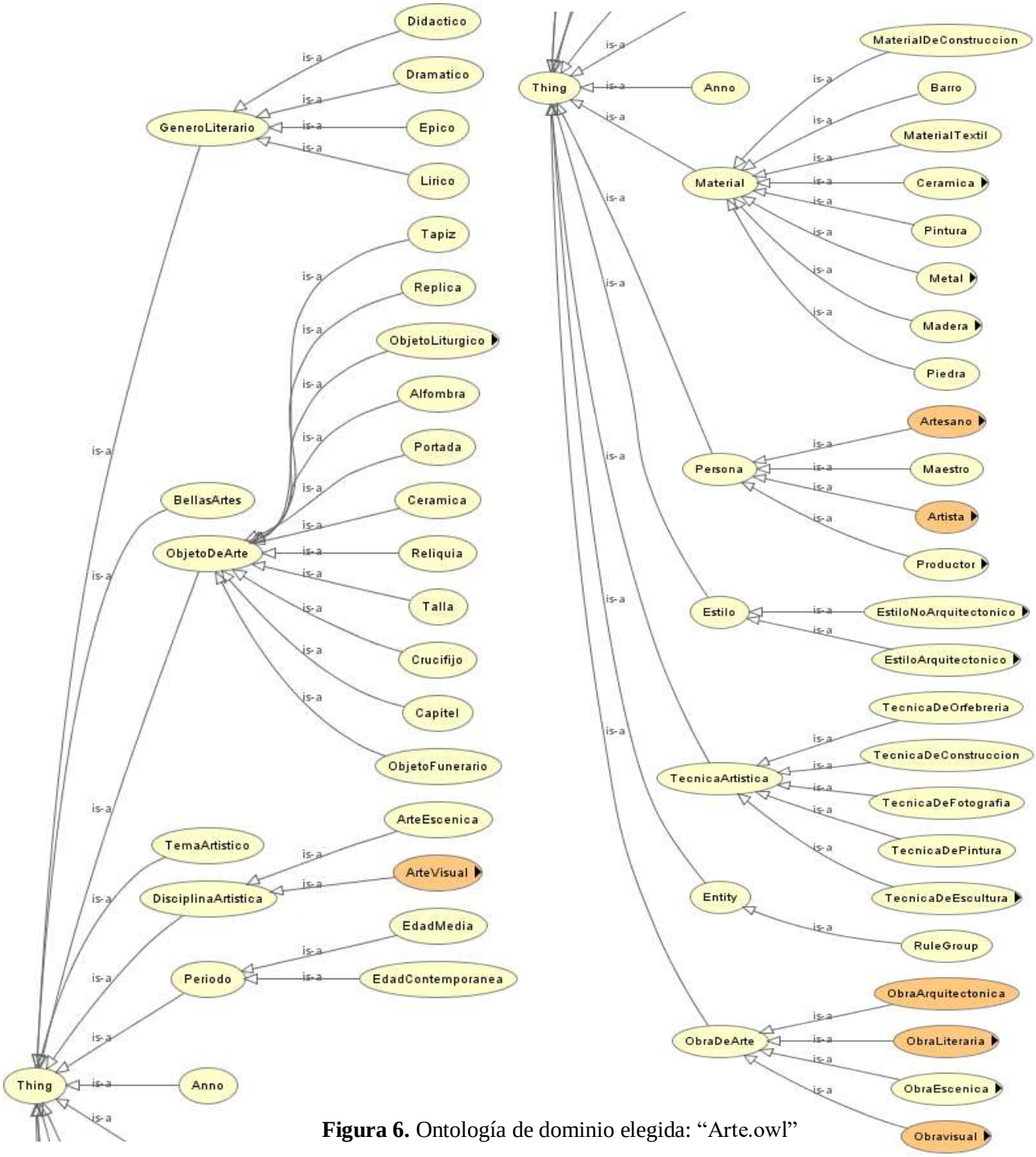

Figura 6. Ontología de dominio elegida: "Arte.owl"

La ontología de Arte sirvió para incorporar la gramática asociada al contexto del Museo de Arte religioso de la ciudad de Popayán, contexto o dominio elegido para la realización de las pruebas mostradas en el presente capítulo y en la aplicación móvil basada en voz del piloto presentado en el capítulo 4. Por consiguiente, se crearon nuevas instancias correspondientes a las obras u objetos de arte, autores y técnicas artísticas del museo.

La Ontología Arte.owl es una de las ontologías de dominio que conforman el conocimiento del Camino de Santiago[1] escrita en castellano, formada por distintos módulos ontológicos[2], cuyo desarrollo siguió la metodología NeOn [120], utilizando como herramienta de construcción de ontologías Protégé 4 y OWL-DL como lenguaje de implementación para dicha ontología. Arte.owl pertenece al proyecto [121], cuyo objetivo era el desarrollo de una red de ontologías basada en Lógica Descriptiva que representara el conocimiento relacionado con el Camino de Santiago, el cuál involucra dominios tan diversos como el arte, la geografía, la arquitectura, entre otros.

El objetivo de esta red de ontologías es permitir el acceso y la edición de información referente al Camino de Santiago para el proyecto internacional GeoBuddies[3].

La manera como se evaluó la ontología de dominio Arte.owl, utilizando el esquema de evaluación de [119] fue la siguiente:

3.2.1.1 Fase 1. Uso correcto del lenguaje

Se evaluó la calidad de la ontología considerando la forma como está escrita, del siguiente modo:

Actividad a. La ontología está codificada en el sublenguaje de OWL-DL (criterio importante de escogencia ya que es el estándar recomendado por W3C [122]), el cual permite máxima expresividad sin perder la completitud computacional. La ontología presenta un lenguaje sólido[4] y completo[5]. Por ello, se pudieron aplicar métodos de razonamiento sobre la ontología de manera satisfactoria.

Actividad b. Para comprobar que la escritura estuviera libre de errores o defectos, se utilizó el marco de chequeo que provee el editor Protégé-OWL. Esta funcionalidad permitió corroborar que la ontología escogida no presentaba inconsistencias sintácticas, lo que indica que se está trabajando con un código libre de errores. Adicionalmente se utilizó el analizador sintáctico de archivos OWL de la Universidad de Manchester[6].

3.2.1.2 Fase 2. Exactitud de la estructura taxonómica

Los elementos a considerar son: identificación de inconsistencias, completitud de conceptos y existencia de redundancias en clases, instancias y relaciones. Con esto se logró:

---

[1] Ruta que recorren los peregrinos procedentes de toda España y de toda Europa para llegar a la ciudad de Santiago de Compostela, donde se veneran las reliquias del apóstol Santiago el Mayor.
[2] No es más que una Ontología a la que se le añaden términos de otra superior, para luego importarla como módulo de la misma.
[3] http://www.geobuddies.net/
[4] Cualquier expresión pueda ser derivada a partir del conocimiento codificado
[5] Cualquier expresión que esté lógicamente implícita en la base de conocimiento pueda ser derivada.
[6] http://www.mygrid.org.uk/OWL/Validator

Observar la estructura jerárquica usada para representar el conocimiento y verificar que presentaba una buena jerarquía, esto, al dividir la propiedad y/o atributo en una jerarquía de propiedades y/o atributos definiendo para cada sub-propiedad/sub-atributo el dominio y/o rango concreto. En el dominio y/o rango de la clase padre se observan todas las clases que son dominio y/o rango de alguna propiedad y/o atributo hija, unidas por "OR".Ejm:

Para el Dominio Obravisual. Se encuentra una propiedad que especifica cuáles son las clases o atributos heredados que son rango de dicho dominio, unidas por OR.

OntoloyArte:ObraVisualNoTradicional
Or
OntologyArte:ObraVisualTradicional

Visualizar las definiciones y las propiedades de los conceptos y sus relaciones semánticas. Con ello se pudo comprobar que estaba bien conformada, ya que no se repetían definiciones para conceptos diferentes, y las propiedades de los conceptos y relaciones semánticas presentaban coherencia entre las clases que se representaban con dichas relaciones.

Visualizar las definiciones y las propiedades de las instancias. Con esta observación se pudo comprobar que no existían definiciones que se repitieran y que las propiedades de los conceptos y sus relaciones semánticas presentaran coherencia. *Ejm:*

OntologyArte:esDeEstilo some OntologyEstilos:Estilo
OntologyArte:perteneceADisciplina Only DisciplinasArtisticas:DisciplinaArtistica
OntologyArte:perteneceADisciplina has DisciplinasArtisticas:cine
OntologyArte:perteneceADisciplina max 1
OntologyArte:tieneAutor only PersonasArte:Artista
OntologyArte:tieneGenero some TemaArtistico:TemaArtistico

Visualizar gráficamente para cada concepto el nodo padre y los nodos hijos que poseía. Como se observa en la Figura 7, la jerarquización es muy clara.

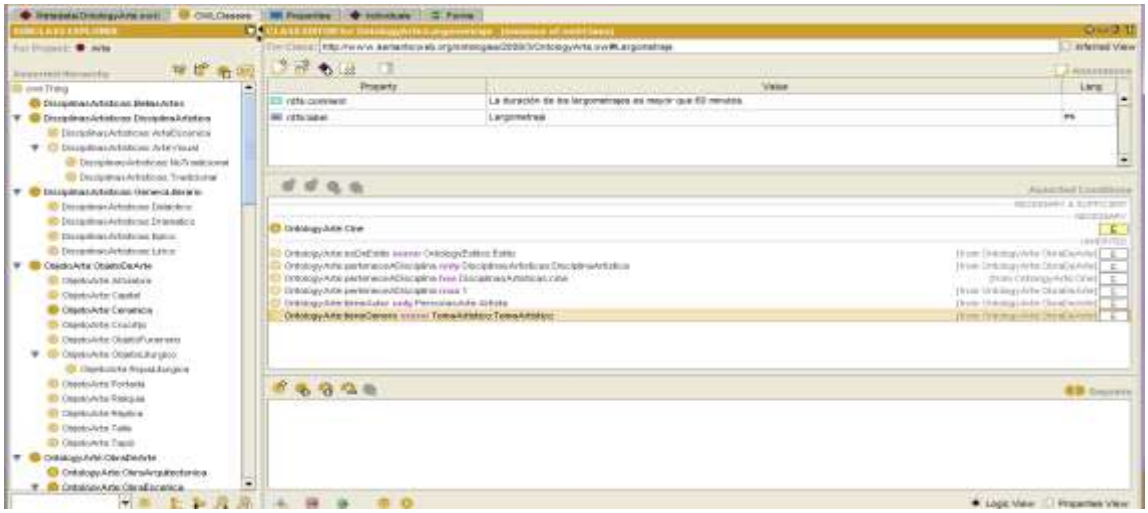

Figura 7. Ontología en el editor Protégé-OWL.

3.2.1.3 Fase 3. Validez del vocabulario.

En esta fase se evalúa el vocabulario usado para describir el conocimiento, utilizando entre otros el *corpus*[7] propio del dominio, construido a partir del texto especializado: "The Art & Architecture Thesaurus" [123], el cual se encuentra especificado en el documento de construcción de la ontología [121] , y los conceptos y definiciones relacionadas con el área del arte, los cuales están disponibles en [124] y [125].

Actividad a. Se identificaron y extrajeron los términos significativos del *corpus*. En los documentos digitales se hizo de manera semiautomática y en los textos se llevó a cabo en forma manual. En total se extrajeron 186 términos que fueron organizados alfabéticamente en una tabla. Por otro lado, el glosario de términos de la ontología contabilizó 172 entradas.

Seguidamente, se contaron la cantidad de términos que se traslaparon entre la ontología y el *corpus*, obteniendo coincidencias para 155 términos.

En resumen se tiene:

CCorp = Cantidad de términos del corpus = 186

COnto = Cantidad de términos de la ontología = 172

CO-C = Cantidad de términos que se solapan entre la ontología y el corpus = 155

Precisión = CO-C / COnto (1)

Actividad b.1. Cálculo de la precisión utilizando la expresión (1)

Precisión = 0,90

Actividad b.2. Cálculo del *recall*, utilizando la expresión (2)

Recall = CO-C / CCorp (2)

Recall = 0,83

Presición	0,90
Recall	0,83

Tabla 1. Resumen de los valores obtenidos para la precisión y el recall de la ontología.

En la Tabla 1 se puede observar que el valor obtenido para la precisión indica que 90% de los términos codificados en la ontología existen en el *corpus*; mientras que el resultado para el *recall* refiere que 83% de los términos del corpus, existen en la ontología.

[7] Conjunto más extenso y ordenado posible de datos o textos científicos, literarios, que pueden servir de base a una investigación (http://rae.es).

Estos porcentajes indican que los conceptos incluidos dentro de la ontología fueron, en un número muy cercano al de los términos del dominio, ya que los términos escogidos por los creadores de la ontología, son altamente especializados y como se indica en [121], el *corpus* proviene de una fuente bastante completa como lo es "*The Art & Architecture Thesaurus"* [92], lo que en el ámbito especializado indica que en la construcción de la ontología se hizo agrupación de términos ordenados jerárquicamente y por disciplinas, permitiendo la denominación normalizada de objetos y/o actividades.

3.2.1.4 Fase 4. Adecuación a requerimientos

En esta fase se verifica y valida que los requerimientos especificados se alcancen de manera satisfactoria.

Actividad a. Esta actividad se realizó para cada fase del ciclo de vida del desarrollo de la ontología, verificando que las especificaciones del documento se alcanzaran [121], haciendo especial énfasis en el cumplimiento de los objetivos, en los formalismos de representación del conocimiento y en la consecución de respuestas correctas para las preguntas de competencia.

Actividad b. Se realizaron recorridos sobre la ontología para verificar que el conocimiento representado permitiera responder las preguntas de competencia, algunas de las cuales fueron:

Consultar todas las obras de arte, obteniendo para cada una su: nombre, técnica artística, material, autor, siglo y género artístico.

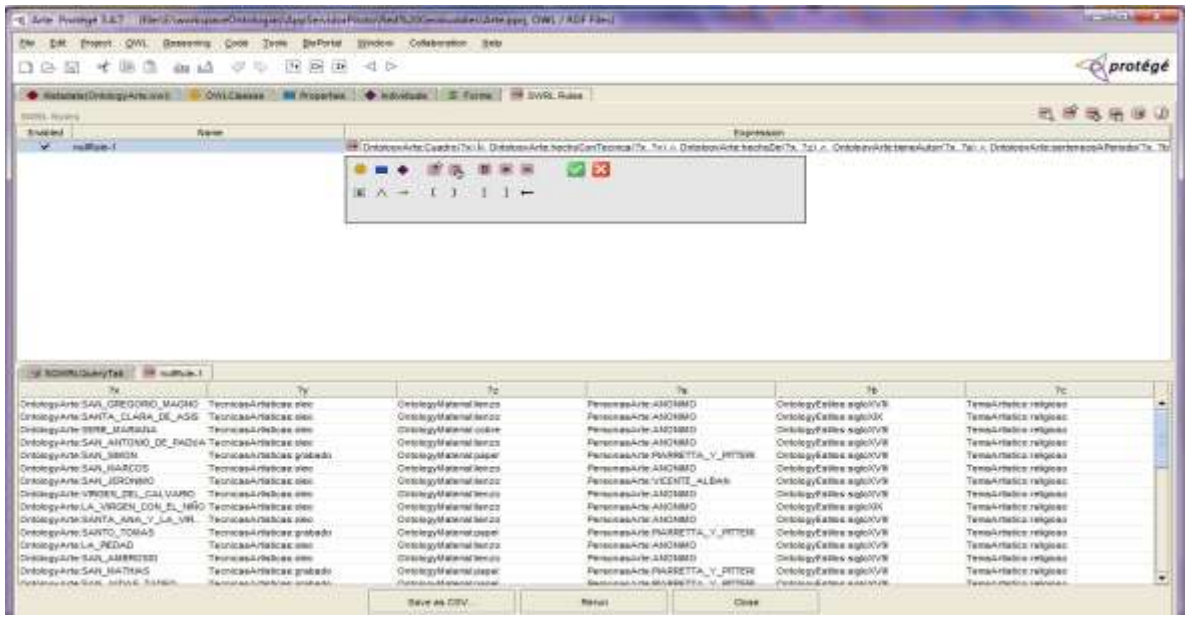

Figura 8. Consulta SQWRL de todas las obras de Arte en el Editor Protégé-OWL

Consultar la obra de arte denominada "ADORACION_CINCO_RAZAS", obteniendo su: nombre, técnica artística, material, autor, siglo y género artístico.

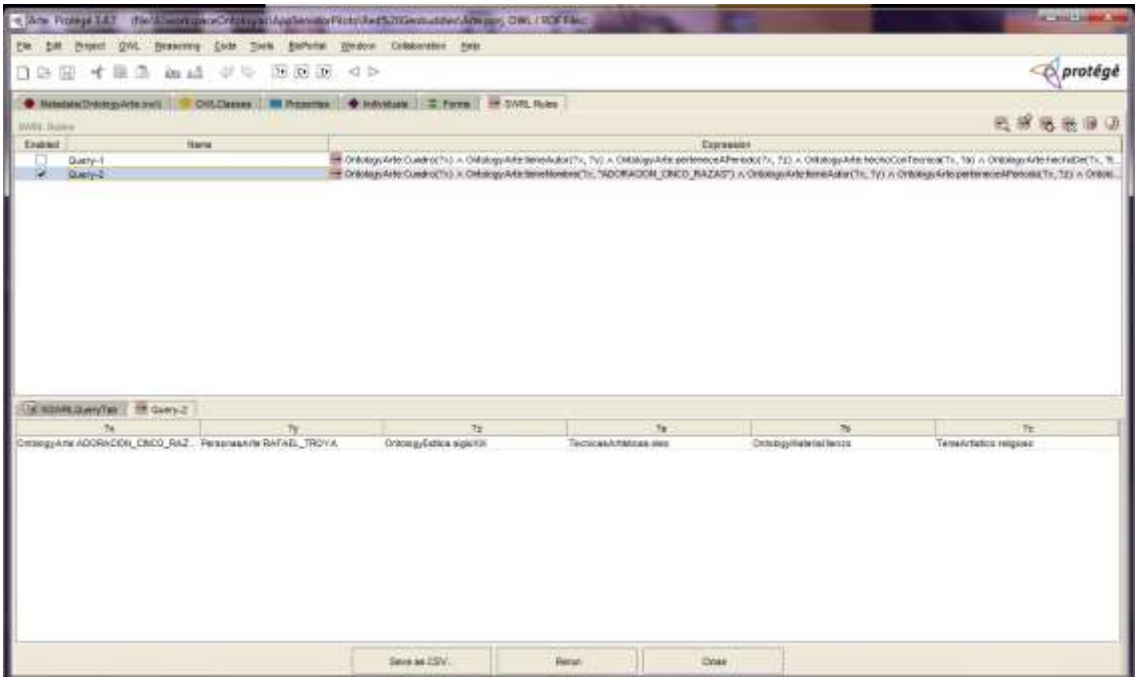

Figura 9. Consulta SQWRL de la obra de Arte denominada "ADORACION_CINCO_RAZAS" en el Editor Protégé-OWL

Los recorridos sobre la estructura taxonómica permitieron encontrar las respuestas a estas preguntas, las cuales son algunas de las que se emplearán en Piloto del Trabajo de Grado, dichas consultas se realizaron en el lenguaje de consultas SQWRL soportado por el motor experto Jess, habilitado en el editor Protégé-OWL.

3.2.2 Conclusiones

La evaluación de la ontología del dominio Arte.owl del Proyecto RED DE ONTOLOGÍAS PARA EL CAMINO DE SANTIAGO [121] fue posible gracias a la aplicación del esquema presentado en [119] y los resultados alcanzados fueron satisfactorios. La aplicación del esquema propuesto para evaluar cada fase del ciclo de vida de desarrollo de la ontología permitió comprobar que la ontología no presentaba errores e inconsistencias sintácticas en el archivo OWL-DL, ni redundancia e inconsistencias para algunas clases e instancias, tampoco omisiones en el vocabulario utilizado para representar el conocimiento.

En la utilización de ontologías creadas por terceros es importante referirse a la documentación para conocerla metodología con la cual fue creada, para establecer así, nuevos criterios que les permitan incrementar la confiabilidad en los resultados provenientes de la evaluación que se realice. Así por ejemplo, para el caso de la ontología Arte.owl, conocer cuál fue la metodología sobre la cual se construyó, es decir *NeOn* [126], de alguna manera permitió garantizar que el desarrollo se había realizado adecuadamente, ya que según la documentación, ésta metodología fue creada con el objeto de cubrir las carencias de las tres metodologías anteriores a ella: METHONTOLOGY, On-To-Knowledge y DILIGEN.

Finalmente se puede decir que una ontología de calidad permite que las tareas y aplicaciones de software que la utilicen o implementen, ofrezcan resultados exitosos y coherentes.

3.3 EVALUACIÓN DE LAS CAPACIDADES DE CONFIGURACIÓN Y RECONOCIMIENTO DE VOZ DE LOS MOTORES ASR

El propósito de la siguiente evaluación es encontrar las facilidades de configuración de la gramática de contexto de los motores de código abierto más utilizados, siendo éstos los de especial interés para los propósitos de este trabajo, ya que la mayoría de trabajos relacionados con tecnologías de reconocimiento de voz emplean soluciones propietarias, y por consiguiente, es justificable someterlos a experimentación para evaluar su rendimiento y desempeño en la plataforma Android.

El cliente sobre el que se realizaron las pruebas fue un dispositivo móvil Nexus S de Samsung, el cual tiene las siguientes especificaciones: 16 GB de memoria interna, 512 MB RAM y Procesador ARM Cortex A8 1GHz.

En este sentido, es necesario definir unos criterios de evaluación para confrontarlos con las características y desempeño de cada motor, determinando y seleccionando los que presenten mejores capacidades de configuración y reconocimiento de voz.

*3.3.*1 Selección de los motores ASR.

De acuerdo al contexto anteriormente planteado, los motores elegidos fueron: PocketSphinx, Julius y la API de reconocimiento de Voz de Android.

- **Características de los motores ASR**

1. **PocketSphinx.**

Es un pequeño siste

ma de reconocimiento continuo del habla, desarrollado por la Universidad de Carnegie-Mellon y se ofrece con licencia BSD de libre distribución, especialmente diseñado para dispositivos portátiles. Depende de una librería llamada SphinxBase, la cual proporciona funcionalidades comunes a todos los proyectos CMUSphinx [53]. Para su configuración en Android en necesario tener instalado SphinxBase, PocketSphinx y Android NDK (Native Development Kit), el cual es una herramienta complementaria del SDK que permite construir aplicaciones de rendimiento crítico en código nativo, es decir C.

2. **Julius.**

Julius está escrito en C puro y tiene poca dependencia de librerías externas, por ende puede funcionar en otras plataformas. Para que una aplicación pueda utilizar Julius, se debe emplear el software de reconocimiento como tal y un archivo de configuración de propiedades. Los idiomas soportados por éste reconocedor solo incluyen el inglés y el japonés [127].

3. API de reconocimiento de Voz de Android:

El sistema Operativo de Android después de su versión 2.1, ofrece la aplicación: “Google Voice Search”, que viene preinstalado en muchos dispositivos Android y disponible también en Android Market, ofreciendo potentes funcionalidades como "búsqueda por voz" y el manejo de la navegabilidad a través del dispositivo como por ejemplo, el comando de voz: “Vaya a”, para dirigirse a una interfaz especifica [18].

*3.3.*2 Definición de criterios de evaluación

A continuación se presentan los criterios definidos para evaluar el rendimiento de los motores de reconocimiento de voz seleccionados anteriormente:

No.	Criterio de evaluación
1	WER (Word Error Rate)
2	WRR (Word Recognition Rate)
3	WCR (Word Correct Rate)
4	Configuración Gramática
5	Formato de la gramática
6	Flexibilidad en la configuración de la gramática
7	Modelos empleados
8	Frecuencia de muestreo, configuración de canal y formato de audio
9	Procesador del reconocimiento (Cliente/Servidor)
10	Soporte para Español
11	Dificultad en la configuración del motor
12	Documentación

Tabla 2. Criterios de Evaluación para los motores de Reconocimiento de Voz

Para el caso de las métricas de referencia, alusivas a los criterios 1,2 y 3 son las más comúnmente usadas en la evaluación de los ASR. Por su parte, la WER (Word Error Rate) es una medida que ha servido para avanzar en la investigación del reconocimiento de voz en los últimos tiempos, derivada de la distancia Levenshtein o distancia de edición, dicha distancia entre dos cadenas, es el número mínimo (o suma ponderada) de las inserciones, supresiones y sustituciones necesarias para transformar una cadena en otra [128]. La WER es la distancia de edición entre una secuencia de palabras de referencia y su transcripción automática:

$$WER = \frac{S+D+I}{Nr} \quad (1)$$

Donde Nr es el total de palabras en la transcripción de referencia, S es el número de palabras sustituidas, D es el número de palabras de referencia eliminadas, I como el número de palabras insertadas en la transcripción automática y que no aparecen en la referencia, y H el número de palabras correctamente reconocidas.

Aunque esta métrica es la más comúnmente empleada como índice de error, se complementa frecuentemente con la tasa de reconocimiento de palabras WRR (Word Recognition Rate):

$$WRR = 1 - WER = \frac{H-I}{Nr} \quad (2)$$

Donde $H = Nr - (S + D)$. Algunas veces, en especial para los sistemas de reconocimiento de palabras aisladas, la tasa correcta de palabras WCR (Word Correct Rate) se utiliza como una métrica de desempeño para el reconocimiento de voz. y no tiene en cuenta los errores de inserción [129]:

$$WCR = \frac{H}{Nr} \quad (3)$$

La inclusión de los criterios 4, 5 y 6, relacionados con el soporte de configuración de gramática, su formato, y la flexibilidad en la configuración de la misma, tiene sentido, ya que según el contexto del problema y objetivos perseguidos por este trabajo de grado, y como se ha mencionado anteriormente, el éxito de la interacción con los sistemas de reconocimiento de voz depende en gran medida de la restricción del lenguaje o vocabulario específico a ser utilizado en un dominio de aplicación concreta, y este lenguaje se define como la gramática de contexto, la cual a su vez, se puede representar por medio de ontologías de dominio. Los factores influyentes en la escogencia de dichos criterios están relacionados con las características intrínsecas de cada motor ASR, ya que los que permiten configuración de gramática tienen su propio lenguaje de escritura, haciéndose necesaria la exploración de dichos lenguajes, así como las recomendaciones estándar existentes de los mismos. Por otro lado, el criterio 6 se refiere a la complejidad que se tenga en el momento de escribir las gramáticas de prueba en la experimentación.

El criterio 7 hace referencia a si efectivamente se configuran los modelos básicos que se deben proporcionar para el reconocimiento de voz en cada motor o si se emplean otra clase de modelos. Dichos modelos deben definir la unidad de reconocimiento, propiedades de audio de cada unidad y la restricción lingüística para la conexión entre las unidades del idioma de destino. Entre los cuales se encuentran:

Diccionario de palabras: define el vocabulario o palabras a ser reconocidas y sus pronunciaciones como una secuencia de fonemas.

Modelo de lenguaje, define las reglas de nivel sintaxis que define la restricción de conexión entre las palabras. Puede ser una gramática basada en reglas o un modelo probabilístico de palabras N-gram.

El modelo acústico, es un modelo estocástico de los patrones de onda de entrada, por lo general por fonema [130].

El criterio 8 considera los tres parámetros importantes que deben ser fijados en el formato de audio que utilizan los motores ASR, los cuales en muchos casos se encuentran pre-establecidos: la frecuencia de muestreo, bits por muestra y número de canales (monofónico o estéreo). Por su parte, la frecuencia de muestreo debe ser compatible con el dispositivo y debe ser la misma con la que ha sido grabado el modelo acústico que se va a utilizar para el reconocimiento [130].

Finalmente, en los criterios 9, 10, 11 y 12 se encuentran: la ubicación del motor de reconocimiento (Cliente/Servidor), los problemas presentados en la configuración de los motores, la documentación encontrada para resolver dichos problemas, y el soporte del idioma español.

*3.3.*3 Plan de Pruebas y evaluación.

Para evaluar las capacidades de configuración y reconocimiento de voz de los motores ASR previamente seleccionados, se calculan las métricas WER, WRR y WCR empleando las ecuaciones (1), (2) y (3) respectivamente; los motores se someten a un plan de pruebas consistente de 10 iteraciones para cada palabra o frase de referencia, correspondientes a la instanciación de una ontología de dominio específica de prueba ("Arte.owl"), y el registro del número de inserciones, supresiones y sustituciones por cada una. Para el caso de Julius, debido a su restricción en los lenguajes soportados, la gramática fue traducida al idioma inglés. Los resultados obtenidos se muestran en las siguientes tablas:

1. **PocketSphinx**

Palabra o Frase de referencia	Numero Sustituciones (D)	Numero Supresiones (S)	Numero Inserciones (I)	Numero Correctas (H)	WER	WRR	WCR
OBRA DE ARTE	0	0	0	10	0	1	1
ARQUITECTONICA	1	0	0	9	0,25	0,9	0,9
VISUAL	1	1	0	8	0,2	0,8	0,8
LITERARIA	0	0	0	10	0	1	1
ESCENICA	2	0	0	8	0,2	0,8	0,8
DRAMATICA	2	0	0	8	0,2	0,8	0,8
EPICA	5	0	0	5	0,5	0,5	0,5
DIDACTICA	1	0	0	9	0,1	0,9	0,9
LIRICA	0	0	0	10	0	1	1
ORATORIA	0	0	0	10	0	1	1
ROMANCE	0	0	0	10	0	1	1
POEMA EPICO	3	0	0	7	0,3	0,7	0,7
NOVELA	0	0	0	10	0	1	1
CUENTO	0	0	0	10	0	1	1
EPOPEYA	5	0	0	5	0,5	0,5	0,5
OBRA DE ARTE ARQUITECTONICA	0	1	0	9	0,1	0,9	0,9
OBRA DE ARTE VISUAL	1	0	0	9	0,1	0,9	0,9
OBRA DE ARTE LITERARIA	1	0	0	9	0,1	0,9	0,9
OBRA DE ARTE ESCENICA	1	0	0	9	0,1	0,9	0,9
OBRA DE ARTE LITERARIA DRAMATICA	1	0	0	9	0,1	0,9	0,9
OBRA DE ARTE LITERARIA DIDACTICA	1	0	0	9	0,1	0,9	0,9
OBRA DE ARTE LITERARIA LIRICA	2	0	0	8	0,2	0,8	0,8
OBRA DE ARTE LITERARIA ORATORIA	0	0	0	10	0	1	1
OBRA DE ARTE LITERARIA EPICA	0	0	0	10	0	1	1

Palabra o Frase de referencia	Numero Sustituciones (D)	Numero Supresiones (S)	Numero Inserciones (I)	Numero Correctas (H)	WER	WRR	WCR
ROMANCE							
OBRA DE ARTE LITERARIA EPICA POEMA EPICO	0	0	0	10	0	1	1
OBRA DE ARTE LITERARIA EPICA NOVELA	0	0	0	10	0	1	1
OBRA DE ARTE LITERARIA EPICA CUENTO	0	0	0	10	0	1	1
OBRA DE ARTE LITERARIA EPICA EPOPEYA	0	0	0	10	0	1	1

Tabla 3. Resultados obtenidos en la evaluación del reconocimiento de voz del Motor PocketSphinx

WER Promedio	WRR Promedio	WRC Promedio
0,1089	0,8964	0,8964

Tabla 4. Resultados de la WER, WRR y WRC promedio obtenidos.

El resultado obtenido señala que de 10 iteraciones por cada palabra, en promedio, sólo el 10% fueron reconocidas erróneamente. Se puede observar igualdad entre la WRR y WCR, debido a que no existe ninguna inserción en las 10 iteraciones por la configuración de la gramática, ya que el reconocedor no inserta palabras que no existan en la misma.

2. Julius

Entre los lenguajes soportados por Julius no se encuentra el español, es por este motivo, que la gramática a configurar se debe traducir al idioma inglés para efectuar las pruebas de reconocimiento.

Palabra o Frase de referencia	Numero Sustituciones (D)	Numero Supresiones (S)	Numero Inserciones (I)	Numero Correctas (H)	WER	WRR	WCR
WORK OF ART	10	0	0	0	1	0	0
ARCHITECTURAL	8	0	0	2	0,8	0,2	0,2
VISUAL	10	0	0	0	1	0	0
LITERARY	10	0	0	0	1	0	0
SCENIC	0	0	0	10	0	1	1
DRAMA	3	0	0	7	0,3	0,7	0,7
EPIC	0	0	0	10	0	1	1
LIRYC	8	0	0	2	0,8	0,2	0,2
ORATORY	9	0	0	1	0	1	1
ROMANCE	0	0	0	10	0	1	1
EPIC POEM	3	0	0	7	0,3	0,7	0,7
NOVEL	9	0	0	1	0,9	0,1	0,1
STORY	6	0	0	4	0,6	0,4	0,4
SAGA	7	0	0	3	0,7	0,3	0,3

Tabla 5. Resultados obtenidos en la evaluación del reconocimiento de voz del Motor Julius

WER Promedio	WRR Promedio	WRC Promedio
0,5285	0,4714	0,4714

Tabla 6. Resultados de la WER, WRR y WRC promedio obtenidos.

El anterior resultado indica que de 10 iteraciones por cada palabra, en promedio un 50% son erróneas. La igualdad entre la WRR y WCR es justificable ya que en las 10 iteraciones no hubo ninguna inserción por parte del reconocedor, y éste no inserta palabras que no existan en la gramática configurada.

De acuerdo a lo anterior se observa que la tasa correcta de reconocimiento de voz es equivalente a la tasa de error, es decir aproximadamente 50%, señalando un desempeño poco eficiente para el reconocedor en estudio. La limitante impuesta por el reconocedor en el soporte del idioma español, hizo que la gramática fuera traducida al idioma inglés, ocasionando mayor dificultad en el reconocimiento.

3. API de reconocimiento de Voz de Android:

Palabra o Frase de referencia	**Numero Sustituciones (D)**	**Numero Supresiones (S)**	**Numero Inserciones (I)**	**Numero Correctas (H)**	**WER**	**WRR**	**WCR**
OBRA DE ARTE	1	0	0	9	0.1	0.9	0.9
ARQUITECTONICA	0	0	0	10	0	1	1
VISUAL	0	0	0	10	0	1	1
LITERARIA	1	0	0	9	0.1	0.9	0.9
ESCENICA	2	0	0	8	0.2	0.8	0.8
DRAMATICA	9	0	0	1	0.9	0.1	0.1
EPICA	1	0	0	9	0.1	0.9	0.9
DIDACTICA	0	0	0	10	0	1	1
LIRICA	6	0	0	4	0.6	0.4	0.4
ORATORIA	0	0	0	10	0	1	1
ROMANCE	2	0	0	8	0.2	0.8	0.8
POEMA EPICO	0	0	0	10	0	1	1
NOVELA	0	0	0	10	0	1	1
CUENTO	1	0	0	9	0.1	0.9	0.9
EPOPEYA	0	0	0	10	0	1	1
OBRA DE ARTE ARQUITECTONICA	1	1	0	8	0.2	0.8	0.8
OBRA DE ARTE VISUAL	1	0	0	9	0.1	0.9	0.9
OBRA DE ARTE LITERARIA	3	0	0	7	0.3	0.7	0.7
OBRA DE ARTE ESCENICA	8	0	0	2	0.8	0.2	0.2
OBRA DE ARTE LITERARIA DRAMATICA	5	0	0	5	0.5	0.5	0.5
OBRA DE ARTE LITERARIA DIDACTICA	4	0	0	6	0.4	0.6	0.6
OBRA DE ARTE LITERARIA LIRICA	2	0	0	8	0.2	0.8	0.8

Palabra o Frase de referencia	Numero Sustituciones (D)	Numero Supresiones (S)	Numero Inserciones (I)	Numero Correctas (H)	WER	WRR	WCR
OBRA DE ARTE LITERARIA ORATORIA	3	0	0	7	0.3	0.7	0.7
OBRA DE ARTE LITERARIA EPICA ROMANCE	0	0	0	10	0	1	1
OBRA DE ARTE LITERARIA EPICA POEMA EPICO	5	0	0	5	0.5	0.5	0.5
OBRA DE ARTE LITERARIA EPICA NOVELA	3	0	0	7	0.3	0.7	0.7
OBRA DE ARTE LITERARIA EPICA CUENTO	2	0	0	8	0.2	0.8	0.8
OBRA DE ARTE LITERARIA EPICA EPOPEYA	3	0	0	7	0.3	0.7	0.7

Tabla 7. Resultados obtenidos en la evaluación del reconocimiento de voz del Motor API de Reconocimiento de Voz Android

WER Promedio	WRR Promedio	WRC Promedio
0.2246	0,7714	0,7714

Tabla 8. Resultados de la WER, WRR y WRC promedio obtenidos.

El resultado anterior indica que de 10 iteraciones por cada palabra, un 20% se reconocen erróneamente. La métrica WRR es igual a la WCR, ya que no hay gramática de referencia con la cual comparar, por lo tanto solo puede haber sustituciones o eliminaciones, pero no inserciones de palabras en las 10 iteraciones.

En la Tabla 8 se observa que la tasa correcta de reconocimiento de voz es mayor que la tasa de error, es decir un 75% aproximadamente de palabras correctamente reconocidas, mostrando un desempeño aceptable del reconocedor en estudio, no obstante, la falta de configuración de una gramática de referencia se evidencia en la ambigüedad en el reconocimiento.

*3.3.*4 Configuración de los motores ASR.

Para la configuración de la gramática en cada uno de los motores de reconocimiento de voz sobre Android, se debe tener en cuenta los siguientes aspectos:

1. **PocketSphinx**

Para PocketSphinx se debe obtener el Modelo Acústico, el modelo de lenguaje o gramática y el diccionario de pronunciación, los cuales deben ser configurados adecuadamente en la aplicación de Android. Como se mencionó anteriormente, dicho modelo acústico está basado en los Modelos Ocultos de Markov (HMM), y con las herramientas necesarias, puede construirse y entrenarse un modelo propio o descargar alguno ya existente para el idioma requerido. Como la creación y entrenamiento de un modelo acústico es una tarea bastante dispendiosa y no se encuentra dentro del

alcance del trabajo de grado, se descargó el Modelo Acústico para la Lengua Española disponible en la página de CMU Sphinx [131], el cual es apto para el idioma español, y se encontraba grabado a 8kHz, por tanto es necesario un canal configurado monofónico, con 16 bits por muestra [55, 57] y frecuencia de muestreo de 8kHz.

Por otro lado, en vez de configurar un modelo de lenguaje se puede configurar un archivo de gramática, y PocketSphinx adopta el formato JSGF (JSpeech Grammar Format) para su escritura. JSGF es un formato de escritura de gramática para el reconocimiento de voz, es independiente de la plataforma y está basado en reglas gramaticales (conocidas como gramáticas de comando y control) [57]. Dentro del archivo ".jsgf" se configuran ciertas reglas de gramática con palabras específicas las cuales fueron particularmente las que aparecen en las tablas del plan de pruebas como palabras o frases de referencia. De igual manera, el diccionario de pronunciación es un archivo ".dict", el cual es necesario modificar ya que todas las palabras incluidas en la gramática deben estar en el diccionario y tener su respectiva pronunciación. Para la realización de las pruebas se incluyeron las palabras de la gramática que no estaban presentes en el diccionario, junto con su respectiva pronunciación.

2. **Julius**

En Julius se deben preparar "modelos" para el idioma de destino, es decir, es necesario un diccionario de Palabras, un modelo de lenguaje y un Modelo acústico [130]. Para éste último se empleó el modelo ejemplo en inglés disponible en la página de Julius, ya que como se mencionó en el anterior ítem, dentro del alcance del trabajo no se encuentra la creación de éste modelo. Julius emplea 16 bits por muestra, ya que no soporta 8 o 24 bits, el número de canales debe ser uno, porque el dispositivo debe soportar el canal de grabación, y finalmente la frecuencia de muestreo por defecto es 16.000Hz, la misma a la que se encontraba grabado el modelo acústico descargado.

Por otra parte, en vez de un modelo de lenguaje, Julius puede llevar a cabo el reconocimiento de voz basándose en una gramática escrita; dicha gramática describe la posible sintaxis o patrones de palabras en una tarea específica. La gramática que se debe reconocer debe ir en dos archivos separados: ".grammar" y ".voca". El archivo ".grammar" define la sintaxis a nivel de categoría, es decir, permite la conexión de palabras por su nombre de categoría, y el archivo ".voca" define las palabras candidatas en cada categoría, con la información de su pronunciación. Una vez se configura la gramática con las palabras o frases de referencia que aparecen en las tablas del plan de pruebas traducidas al inglés, ambos archivos son convertidos a archivos ".dfa" y ".dict" por medio del compilador de gramática "mkdfa.pl" de Julius [132].

3. **API de Reconocimiento de Android:**

Android es una plataforma abierta, por lo que la aplicación puede hacer uso de cualquier servicio de reconocimiento de voz instalado en el dispositivo, para que reciba un objeto de la clase RecognizerIntent.

Es importante definir qué tipo de dictado se va a realizar en el método intent.putExtra(String, String). Si se desea un dictado libre, se debe fijar LANGUAGE_MODEL_FREE_FORM como segundo parámetro, y si se desea hacer la búsqueda web con mensajes cortos, entonces se debe fijar LANGUAGE_MODEL_WEB_SEARCH para el mismo.

En este sentido, se tiene la posibilidad de utilizar dos modelos de lenguaje: búsqueda en la Web para palabras cortas y el modelo de forma libre para dictados, los cuales están disponibles en varios idiomas. Para lograr un cierto grado de funcionalidad "básica" con Google Voice Searh, Google creó tres tipos de reconocimiento, uno basado en la forma de utilización de lenguaje en México, otro para Argentina y el último, para todos los demás países y regiones de Latinoamérica. Se recolectaron datos en distintas condiciones acústicas, como restaurantes, en vías públicas o en automóviles en movimiento tan sólo por citar algunos ejemplos, para permitir que el modelo de reconocimiento de voz pudiese ser utilizado en tantos escenarios de uso como fuese posible [133]. Si bien es posible realizar configuraciones de la escogencia del idioma, no es posible específicamente realizar la configuración de gramática de contexto, y esto debido al procesamiento remoto que realiza la plataforma de Android para cualquier palabra capturada por el API de reconocimiento de voz. Los diccionarios de palabras para cada idioma soportado se encuentran almacenados en los servidores de Google, lo que impide lógicamente configurar una gramática propia.

3.3.5 Resultados y conclusiones

En la Tabla 9. Resultados de Evaluación Motores de Reconocimiento de VozTabla 9 se presenta un resumen con los resultados de la evaluación de los motores de reconocimiento de voz seleccionados. En este sentido, las conclusiones se presentan de acuerdo a aspectos relacionados con las facilidades o restricciones que traen consigo la configuración y el reconocimiento de voz de los motores ASR, y que determinan finalmente, la escogencia de uno de ellos:

El motor de reconocimiento Julius permitió configurar una gramática a reconocer. Sin embargo, debido a la limitante impuesta por el soporte del idioma español, soportando sólo el inglés y el japonés, estuvo constituida por palabras en inglés, las cuales correspondieron a la traducción de las palabras y frases de referencia obtenidas de la ontología de dominio elegida. De acuerdo a lo anterior, se presentó mayor dificultad en el reconocimiento de las frases, en comparación con los otros motores que si lo permitieron realizar en español.

Cuando se incluyó el API de reconocimiento de Android en la aplicación, se configuraron los parámetros relacionados con el reconocimiento mencionados anteriormente. No obstante, la configuración es muy limitada, ya que a pesar del amplio soporte de idiomas y la cantidad de palabras a reconocer (sobre todo la riqueza del léxico latinoamericano), no fué posible configurar una gramática específica, generando ambigüedad el reconocimiento de varias consultas realizadas, ya que cada palabra capturada por el API es enviada a los servidores de Google directamente para su respectivo procesamiento, en donde se encuentra la base de datos con los diccionarios de los idiomas soportados.

Con el motor de reconocimiento PocketSphinx se pudo realizar la configuración adecuada de la gramática a reconocer, constituida por las palabras y frases de referencia obtenidas de la ontología de domino elegida en español, siendo más eficiente en el reconocimiento de las mismas. Igualmente, una de las características de PocketSphinx es su estabilidad en términos de fuente y compatibilidad binaria, debido a la utilización de tipos abstractos, lo cual permite una modesta pero significativa reducción en el consumo de memoria, haciéndolo apto para dispositivos móviles.

Motor	WER	WRR	WCR	Configuración Gramática	Formato de la gramática	Flexibilidad en la configuración de la gramática	Uso de modelos	Frecuencia de muestreo, configuración de canal y formato de audio	Procesador del reconocimiento (Cliente/Servidor)	Soporte para Español	Dificultad en la configuración del motor	Documentación
Pocket Sphinx	0,1089	0,8964	0,8964	Si	JSGF Estándar	Media, la gramática se define en un solo archivo .jsgf, un formato basado en reglas de gramática.	Modelo Acústico, modelo de lenguaje o gramática y diccionario de pronunciación	8000 Hz, MONO, PCM 16 Bits	Dispositivo	Si	Media	Alta
Julius	0,5285	0,4714		Si	BNF-like	Media, requiere dos archivos: un .grammar que define la categoría a nivel de sintaxis y un .voca, con las palabras de cada categoría y su pronunciación.	Modelo Acústico, modelo de lenguaje o gramática y diccionario de pronunciación	16000 Hz, MONO, PCM 16 Bits	Servidor	No	Media	Media
API de reconocimiento de voz de Android	0,2246	0,7714	0,7714	No	No aplica.	No aplica, No existe posibilidades de configuración de la gramática	Dos modelos de lenguaje: free_form para dictado, y web_search para palabras cortas.	Dependen del dispositivo.	Servidor	Si	Baja	Alta

Tabla 9. Resultados de Evaluación Motores de Reconocimiento de Voz

3.4 Evaluación de los componentes de razonamiento ontológico.

Las herramientas que conforman los componentes de razonamiento ontológico que son de especial interés para los propósitos de este trabajo, son de código abierto, y para su evaluación es necesario definir unos criterios de evaluación para confrontarlos con sus características y eficiencia en inferencias de conocimiento, con el ánimo de determinar y seleccionar un mecanismo adecuado que procese la gramática de contexto de la aplicación.

3.4.1 Descripción de los componentes para el razonamiento ontológico.

El propósito de la siguiente evaluación es seleccionar un mecanismo que procese la gramática de contexto de la aplicación. Es por esto, que a partir del estado del arte realizado y la exploración de los trabajos relacionados con el razonamiento ontológico [113, 116, 134-136], se concluyó que se podían combinar diversos componentes prevalecientes en uno u otro trabajo, los cuales permitieran un adecuado procesamiento de la ontología y suplieran las necesidades del trabajo de grado.

De acuerdo al estudio realizado, los componentes necesarios para el razonamiento ontológico son: un razonador, un motor de reglas, un lenguaje de consulta y una API de programación para ontologías. Los razonadores ontológicos permiten generar conocimiento y hacer inferencias a partir de un conjunto de axiomas y hechos, por otra parte la definición de reglas y su ejecución desempeña un papel importante en el proceso de inferencia de conocimiento, debido a que el comportamiento de una instancia dentro de un dominio puede ser expresado a través de reglas o axiomas; por lo tanto, de la buena definición de estas depende el éxito de la generación de nuevo conocimiento [89]. Los lenguajes de consultas proporcionan al usuario la capacidad para recuperar información de la ontología [105], y finalmente el API de programación para ontologías proporciona clases y métodos para crear, cargar y guardar archivos OWL, consultar y manipular modelos de datos y realizar razonamiento a través de los motores [72] [137].

En este sentido, se evaluarán las herramientas de código abierto más comunes que implementan los componentes anteriormente mencionados.

Dado que las tareas de razonamiento ontológico requieren una alta capacidad de procesamiento se dejaron en un servidor Web. En ese sentido, se usó una máquina con las siguientes características para ejecutar el servidor: 320 GB de disco duro, 4GB RAM, Procesador Intel Core 2Duo 2.2GHz y tarjeta inalámbrica 802.11b.

3.4.2 Definición de criterios de evaluación.

A continuación se presentan los criterios para evaluar el rendimiento de las herramientas involucradas en el razonamiento, divididos en las siguientes partes, según los componentes del mecanismo:

- Criterios de Evaluación de los razonadores de lógica descriptiva y de programación lógica más comunes.
- Criterios de Evaluación de los motores de reglas para ontologías más comunes.
- Criterios de Evaluación de los lenguajes de consulta de ontologías sobre los motores más comunes.

3.4.2.1 Definición de los Criterios de Evaluación de los razonadores de lógica descriptiva y de programación lógica más comunes

A continuación se presentan los criterios definidos para evaluar el rendimiento de los razonadores de lógica descriptiva y de programación lógica más comunes:

No.	Criterio de evaluación
1	Algoritmo de razonamiento
2	Verificación de consistencia y clasificación de ontologías
3	Tiempo de validación y clasificación de la ontología
4	Precisiones sobre los conceptos de la jerarquía
5	Lenguaje
6	Vinculación OWL-DL
7	Soporte interfaz DIG
8	Soporte tipos de datos
9	Soporte de consultas
10	Soporte de reglas
11	Documentación y ejemplos
12	Comunidad de desarrolladores implicados
13	Facilidad de instalación

Tabla 10. Criterios de Evaluación de los razonadores de lógica descriptiva y de programación lógica más comunes

En cuanto al primer criterio, éste se encuentra relacionado con el algoritmo de razonamiento de las distintas implementaciones para razonar con los diferentes formalismos lógicos: Lógica de Primer Orden, Lógicas Descriptivas y Programación Lógica [138], ya que éste varía de acuerdo a dichos formalismos. Cada uno de estos formalismos provee diversos niveles de expresividad referente a la complejidad de cálculo para inferir nuevo conocimiento aplicando distintas estrategias de inferencia [76]. Específicamente, en este trabajo de grado se van a evaluar los razonadores de lógica descriptiva y de programación lógica, ya que son los que se han usado en los últimos años [76], como se puede observar en los trabajos relacionados con el razonamiento de ontologías [113, 116, 134-136].

En el mismo sentido, las lógicas descriptivas fueron inicialmente llamadas lenguajes conceptuales [80] y permiten representar una base de conocimiento que describen un dominio particular [139], y entre los algoritmos que emplean este tipo de formalismo se encuentran los algoritmos estructurales por subsunción y los basados en *Tableaux* [76]. Por su parte, la programación lógica permite escribir reglas de la forma "si A entonces B", aunque su semántica formal no está basada en interpretaciones de primer orden [138]. Entre los algoritmos que utilizan este tipo de formalismo están: el algoritmo Novel, algoritmos basados en reglas, entre otros [89].

Para la escogencia de los criterios 2, 3 y 4 se consideraron los servicios de inferencia que deben proporcionar los razonadores DL [79-81], y el tiempo que toma cada razonador en efectuarlos:

Validación de la consistencia y cumplimiento de conceptos de una ontología, Clasificación de la ontología y Precisiones sobre los conceptos de la jerarquía descritos en el capítulo 2.

Por otro lado, el criterio 5, correspondiente al lenguaje de programación es importante ya que puede afectar la interacción entre los componentes del mecanismo, y su conocimiento es de vital importancia para establecer compatibilidades entre las herramientas y posibles dificultades de integración.

Por su parte, el criterio 6 está relacionado con la vinculación OWL-DL y se refiere al soporte del lenguaje OWL DL, que proporciona máxima expresividad y exige completitud computacional (se garantiza que todas las conclusiones son computables) y decibilidad (todos los cálculos acaban en un tiempo finito) [89]; su importancia radica en el hecho que la ontología elegida está basada en Lógica Descriptiva.

Para el criterio 7 se consideró el soporte a la interfaz DIG. Esta interfaz permite tener la conexión entre razonadores de DL y editores o API's de Programación de ontologías; para ello utiliza un protocolo basado en HTTP PUT/GET junto con XML esquema, que proporciona un conjunto mínimo de operaciones (por ejemplo, comprobar la subsunción o realizar razonamiento de clasificación) que han demostrado ser útiles en las aplicaciones [140].

En relación al criterio 8, éste corresponde al soporte de tipos de datos de los razonadores ya que algunos tienen ciertas restricciones con algunos de ellos, especialmente para las propiedades especificadas en una ontología, y por ende no las razonan adecuadamente.

Los criterios 9 y 10 especifican el soporte de consultas y de reglas, ya que las consultas proporcionan la capacidad de recuperar información de la ontología [105], y la definición de reglas juega un papel importante en los procesos de inferencia: de la buena definición de las reglas depende el éxito para la generación de nuevo conocimiento [90]. Ambos procesos se realizan por medio de lenguajes de consultas y lenguajes de reglas soportados sólo por ciertos razonadores.

Finalmente, los últimos criterios de evaluación 11, 12 y 13, están ligados a la experimentación e implementación de los prototipos de prueba, ya que algunos razonadores no cuentan con un soporte documental o de apoyo en red adecuado. Adicionalmente, la facilidad de instalación depende de cada motor de razonamiento y los requisitos necesarios para la misma.

3.4.2.2 Definición de los Criterios de Evaluación de los motores de reglas más comunes.

A continuación se presentan los criterios definidos para evaluar el rendimiento de los motores de reglas más comunes:

No.	Criterio de evaluación
1	Método de razonamiento
2	Sintaxis del lenguaje de reglas adoptado
3	Inclusión de reglas en código
4	Axiomas inferidos de reglas establecidas

Tabla 11. Criterios de Evaluación de los motores de reglas más comunes.

El primer criterio está relacionado con el método de razonamiento empleado por el motor de inferencia del motor de reglas, y es el componente más importante, ya que es el encargado de las operaciones de búsqueda y selección de las reglas a utilizar en el proceso de razonamiento [141].

Por su parte, el criterio 2 se refiere a la habilidad del programa o aplicación desarrollada para controlar la estimación y selección de las alternativas mientras éste se ejecuta, dependen directamente de las reglas definidas en la ontología y del lenguaje de reglas utilizado. Los lenguajes de reglas permiten complementar a los mecanismos ya existentes en OWL para realizar las definiciones de información de gestión, de forma que permiten expresar restricciones complejas; valores que dependen de otras variables, relaciones entre objetos, comportamiento de máquina de estado de valores, entre otros [142].

El criterio 3 hace referencia a la facilidad para la inclusión de reglas que se han creado con editores externos dentro de una aplicación propia, como por ejemplo el Editor Protégé-OWL.

Finalmente, el criterio 4 se estableció para determinar la capacidad del motor de reglas para inferir nuevo conocimiento a partir de las reglas establecidas [143].

3.4.2.3 Definición de los Criterios de Evaluación de los lenguajes de consulta de ontologías sobre los motores más comunes.

A continuación se presentan los criterios definidos para evaluar el rendimiento de los lenguajes de consulta de ontologías sobre los motores más comunes:

No.	Criterio de evaluación
1	Estándar al que es dirigido
2	Lenguaje de origen
3	Herramientas que lo soportan
4	Instancias obtenidas en consultas

Tabla 12. Criterios de Evaluación de los lenguajes de consulta de ontologías sobre los motores más comunes.

Para el primer criterio se consideró el estándar de creación de ontologías para el que ha sido propuesto el lenguaje de consultas [144], ya que algunos se especializan en RDF, mientras que otros son dirigidos a OWL, y en éste último se encuentra creada la ontología escogida.

En cuanto al segundo criterio, éste hace referencia al lenguaje de origen o predecesor [144], ya que de éste depende su extensión y expresividad en la ejecución de las consultas.

Por otro lado, el criterio 3 está relacionado con el motor de ejecución del lenguaje de consulta, el cual permite determinar la compatibilidad entre las herramientas a utilizar en el trabajo de grado.

Para finalizar, el último criterio está dirigido al número de instancias que se obtienen una vez se ha procesado las consultas propuestas en los diferentes lenguajes.

3.4.3 Plan de Pruebas y evaluación.

La escogencia del método de razonamiento ontológico más adecuado para la ontología que representa la gramática de contexto de un reconocedor de voz, se basó en la evaluación de los 3 componentes requeridos, mostrando sus respectivos resultados y conclusiones. Cada evaluación se realizó sobre las dos APIs de programación elegidas: Jena y Protégé OWL API.

3.4.3.1 Evaluación de los razonadores más comunes.

Los razonadores de código abierto más comunes sometidos a la evaluación fueron: Pellet, Fact++, Subsistema de Razonamiento de Jena y Racer Pro.
Para la realización de pruebas de rendimiento, se desarrollaron seis prototipos en Java, tres sobre el API de Jena y tres sobre Protégé OWL API. Para el caso de Jena, se emplearon los razonadores Racer Pro y Fact++ por medio de la interfaz DIG sobre HTTP; Pellet con la interfaz DIG directa, y el Subsistema de razonamiento con los métodos propios del API. Por su parte, en el API de Protégé se emplearon los razonadores FaCT++, Racer y Pellet por medio de la interfaz DIG de Protégé sobre HTTP.

Se realizaron varias pruebas de rendimiento, midiendo los tiempos de respuesta de validación de consistencia, clasificación y precisiones sobre conceptos de la ontología como variable de rendimiento, por medio de 5 pruebas iguales por cada razonador, de forma alternada, y reseteando su memoria entre experimento y experimento. Se utilizó la ontología: "Arte.owl", con un peso de 80 KB.

- **Pruebas en Jena API**

Razonador	**1 Prueba (seg)**	**2 Prueba (seg)**	**3 Prueba (seg)**	**4 Prueba (seg)**	**5 Prueba (seg)**	**Media (seg)**
Razonador Genérico Jena	12	17	13	12	11	13
Pellet	1	1	1	1	1	1
Fact++	3	1	1	2	2	1.8
RacerPro	7	9	12	10	8	9.2

Tabla 13. Tiempo de validación de consistencia de la ontología sobre Jena

Razonador	**1 Prueba (seg)**	**2 Prueba (seg)**	**3 Prueba (seg)**	**4 Prueba (seg)**	**5 Prueba (seg)**	**Media (seg)**
Razonador Genérico Jena	28	34	20	20	19	24.2
Pellet	2	3	4	3	2	2.8
Fact++	2	2	1	3	2	2
RacerPro	10	9	15	13	10	11.4

Tabla 14. Tiempo de clasificación de la ontología sobre Jena

Razonador	**1 Prueba (seg)**	**2 Prueba (seg)**	**3 Prueba (seg)**	**4 Prueba (seg)**	**5 Prueba (seg)**	**Media (seg)**
Razonador Genérico Jena	No califica	No califica	No califica	No califica	No califica	No califica
Pellet	1	1	1	1	1	1
Fact++	540	960	660	960	840	792
RacerPro	120	120	120	120	120	120

Tabla 15. Tiempo de precisiones sobre conceptos de la jerarquía de clases sobre Jena

Para probar la precisión sobre conceptos de la jerarquía de clases, se empleó la clase ClassHierarchy que venía incluida en los ejemplos de Jena, la cual tiene como objetivo mostrar toda la jerarquización de clases existente en la ontología y los resultados que se obtuvieron fueron variados. Con el razonador genérico de Jena, no fue posible obtener la jerarquización con esta clase, ya que no calificó para realizar esta prueba, por su limitación de memoria en la carga de la ontología.

- **Pruebas en Protégé OWL API**

Razonador	**1 Prueba (seg)**	**2 Prueba (seg)**	**3 Prueba (seg)**	**4 Prueba (seg)**	**5 Prueba (seg)**	**Media (seg)**
Pellet	2	3	3	2	2	2.4
Fact++	3	2	3	3	2	2.6
RacerPro	2	3	3	3	3	2.8

Tabla 16. Tiempo de validación de consistencia de la ontología sobre Protégé OWL API

Razonador	**1 Prueba (seg)**	**2 Prueba (seg)**	**3 Prueba (seg)**	**4 Prueba (seg)**	**5 Prueba (seg)**	**Media (seg)**
Pellet	2	3	3	3	3	2.8
Fact++	4	3	3	4	3	3.4
RacerPro	3	3	4	4	3	3.4

Tabla 17. Tiempo de clasificación de la ontología sobre Protégé OWL API

Razonador	1 Prueba (seg)	2 Prueba (seg)	3 Prueba (seg)	4 Prueba (seg)	5 Prueba (seg)	Media (seg)
Pellet	4	2	1	1	1	1.8
Fact++	7	7	8	7	8	7.4
RacerPro	5	3	2	2	2	2.8

Tabla 18. Tiempo de precisiones sobre conceptos de la jerarquía de clases sobre Protégé OWL API

Para probar la precisión sobre conceptos de la jerarquía de clases, se emplearon los métodos del API Protégé OWL, con el fin de mostrar la jerarquización de clases existente en la ontología.

- **Resultados y conclusiones:**

	Pellet	**Fact++**	**RacerPro**	**Subsistema de razonamiento de Jena**
Algoritmo de Razonamiento	Tableaux	Tableaux	Tableaux	Basado en reglas
Verificación de consistencia	Si	Si	Sí, excepto nominales.	Incompleto para OWL DL
Lenguaje	Java	C++	LIPS	Java
Vinculación OWL-DL	Si	Si	Si	Incompleto razonador incluido en la distribución estándar
Soporte interfaz DIG	Si	Sí, pero no en últimas	Sí, pero solo para Protégé.	Sí, pero no en últimas versiones

	Pellet	**Fact++**	**RacerPro**	**Subsistema de razonamiento de Jena**
		versiones		
Soporte tipos de datos	Todos	String, integer	Excepto tipos de datos no estándar.	Todos
Soporte de Consultas	Si, en lenguaje SPARQL	No	Si, en su propio lenguaje nRQL	Si, en lenguaje RDQL y SPARQL
Soporte de Reglas	Si	No	Si, en su propio lenguaje.	Si, en su propio lenguaje.
Facilidad de Instalación	Alta	Media	Alta	Alta
Comunidad de desarrolladores implicados	Si	No	No	Si
Documentación y ejemplos	Excelente	Mala	Aceptable	Buena

Tabla 19. Resultados de evaluación de los razonadores de lógica descriptiva y de programación lógica más comunes

Motor	**Tiempo de validación y clasificación de ontología**		**Precisiones sobre los conceptos de la jerarquía**	
	Jena	**Protégé OWL API**	**Jena**	**Protégé OWL API**
Pellet	3 seg.	5 seg.	Buena jerarquización, ya que no repite clases y muestra adecuadamente la jerarquía de clases.	Buena jerarquización, muestra la misma jerarquía de clases que los otros razonadores.
Fact++	4 seg.	6 seg.	Demasiado redundante, en cuanto a la repetición de clases, por ese motivo gasta demasiado tiempo mostrándola.	Buena jerarquización, muestra la misma jerarquía de clases que los otros razonadores.
Subsistema de Razonamiento de Jena	37 seg.	No aplica	No fue posible mostrar la jerarquía de clases en el código.	No aplica.
RacerPro	20 seg.	6 seg.	Algo redundante, en cuanto a la repetición de clases.	Buena jerarquización, muestra la misma jerarquía de clases que los otros razonadores.

Tabla 20. Resultado de medición de tiempos de ejecución de los razonadores sobre Jena y Protégé OWL API

Los resultados indican que en Jena, la jerarquía de clases mostrada por Fact++ es demasiado redundante, en cuanto a la repetición de clases, ya que muestra la misma clase y la misma propiedad demasiadas veces, invirtiendo un tiempo considerable en su despliegue. Por otro lado, Racer Pro muestra un comportamiento similar aunque se nota una menor redundancia en la repetición de clases y por lo tanto, el tiempo transcurrido también es inferior. Finalmente el razonador Pellet, si mostró adecuadamente la jerarquización de clases aunque sin propiedades, no obstante ésto no es relevante para los propósitos del trabajo de grado, siendo además mucho más ágil durante el despliegue. En cuanto al tiempo de ejecución, el motor más rápido es Pellet, seguido de Racer Pro, y finalmente Fact++.

En Protégé OWL API, los resultados que se obtuvieron fueron uniformes, ya que los 3 razonadores mostraron exactamente la misma jerarquía de clases. La diferencia radica en el tiempo que se demoró cada uno en validar, clasificar y mostrar las clases con su respectiva jerarquía; el más rápido fue Pellet, seguido por Racer Pro, y finalmente Fact++.

En resumen, Pellet demostró ser el razonador más rápido usando las dos API, obteniendo resultados uniformes y consistentes en comparación con los otros razonadores. Existe una diferencia entre Pellet y otros razonadores existentes, ya que éste trata las restricciones anónimas definidas en una ontología OWL como expresiones sintácticas y no las devuelve en las respuestas de cualquier consulta, debido a que el tratamiento de cada restricción como una clase hace más difícil el razonamiento de la ontología y los resultados que se obtienen al final no son útiles [145].

3.4.3.2 Evaluación de los motores de reglas más comunes.

Los motores de reglas de código abierto más comunes sometidos a la evaluación fueron: los motores que trae incorporados Jena y Pellet y el motor experto Jess. A la ontología "Arte.owl", se le incorporaron 4 reglas para inferir nuevo conocimiento. En el caso del API de Jena, dada su limitación para incorporar directamente motores de reglas, sólo se realizaron pruebas con el motor de reglas de Pellet y el motor que trae incorporado internamente. Por otro lado, dado el amplio soporte para Jess que brinda el API de Protégé-OWL, solo se realizaron sobre ésta, pruebas con este motor. Las 4 reglas incorporadas indican las siguientes relaciones:

Todas las obras/objetos de arte que pertenecen al siglo XVI son del Estilo "arteDelRenacimiento".
Todas las obras/objetos de arte que pertenecen al siglo XVII son del Estilo "arteBarroco".
Todas las obras/objetos de arte que pertenecen al siglo XVIII son del Estilo "arteBarroco".
Todas las obras/objetos de arte que pertenecen al siglo XIX son del Estilo "romanticismo".

- **Pruebas en Jena API**

Para validar el motor de inferencias basado en reglas de Jena y el razonamiento con reglas SWRL de Pellet se utilizó la ontología Arte.owl con las 4 reglas incorporadas, e importándolas en un mismo repositorio. Para llegar a resultados más precisos, se realizaron 5 pruebas de rendimiento midiendo los tiempos de respuesta de inferencia de los motores y el número de axiomas inferidos sin errores.

Motor de reglas	1 Prueba (seg)	2 Prueba (seg)	3 Prueba (seg)	4 Prueba (seg)	5 Prueba (seg)	Media (seg)
De Pellet	4	3	3	4	3	3,4
De Jena	60	60	55	60	60	59

Tabla 21. Resultado de medición de tiempos de inferencia de los motores de reglas de Pellet y Jena para las 4 reglas definidas.

Motor de reglas	1 Prueba	2 Prueba	3 Prueba	4 Prueba	5 Prueba	Media
De Pellet	15	15	15	15	15	15
De Jena	No aplica	No aplica	No aplica	No aplica	No aplica	No aplica

Tabla 22. Número de axiomas inferidos sin errores por parte de los motores de reglas de Pellet y Jena para las 4 reglas definidas.

Motor de Reglas	Número de axiomas inferidos	Tiempo promedio de inferencia de las reglas
De Pellet	15 y dos con errores.	3,4 seg.
De Jena	No aplica	59 seg.

Tabla 23. Tabla de resultados generales de las pruebas de inferencia por parte de los motores de reglas de Pellet y Jena.

Durante el proceso de prueba se pudo comprobar que el motor de reglas que ofrece Jena, no provee los métodos que permitan visualizar las inferencias que se realizan a partir de las reglas establecidas, y es más lento en el procesamiento de la ontología, adicionalmente no soporta el lenguaje de reglas SWRL, ya que cuenta son su propia sintaxis y no existe editor del mismo; esto obliga a escribir las reglas en un archivo *.rules aparte de la ontología OWL. Pellet por su parte si muestra las declaraciones y el nuevo conocimiento inferido desde las reglas, en este caso, es soportado el lenguaje estándar SWRL, lo que brinda una mayor facilidad de escribir las reglas con el editor Protégé. Igualmente, razona la ontología con las reglas incluidas en ella, y/o especificadas dentro de la clase como tal. En este sentido, es posible concluir que Pellet realiza una inferencia de reglas mucho más completa, aunque no del todo. La Tabla 23 muestra que para 4 reglas definidas, se infieren 15 axiomas y 2 aparecen erróneos, mientras que en el caso del motor de Jena no se puede visualizar la inferencia.

- **Pruebas sobre Protégé OWL API**

Para comprobar la inferencia del motor Jess trabajando sobre el API de Protégé-OWL se añadieron de una en una las 4 reglas a la ontología "Arte.owl", registrando el tiempo en que finalizaba la inferencia. Como punto de referencia se tomó el resultado del total de axiomas inferidos en el Editor Protégé 3.4.7, el cual mostró un total de 16 axiomas inferidos en un tiempo de 650 mseg a partir de las intancias que se crearon.

Para el caso del API de Protégé-OWL, sólo existe el plugin SWRLJessBridge utilizado para incorporar las funcionalidades del componente SWRLTab de Protégé Editor en Java, y que soporta la ejecución de reglas SWRL. Este plugin solo permite crear un puente entre el modelo OWL, incorporando reglas OWL y el motor experto en reglas Jess.

Por su parte, las inferencias en Protégé-OWL utilizando el motor Jess se realizan en tres pasos: un primer paso de traducción de las reglas SWRL y lenguaje OWL a Jess (OWL+SWRL → Jess); un segundo paso de ejecución del motor Jess y obtención de los datos (RunJess); y el tercero que consiste en traducir estos datos generados por Jess a OWL y añadirlos a la ontología (Jess →OWL).

Con 1 Reglas: Axiomas inferidos: 3.

Herramienta	Protégé-OWL API (mseg)
OWL + SWRL→ Jess	650
RunJess	43
Jess → OWL	1
Total	684

Tabla 24. Tabla de resultado de las prueba de inferencia #1 por parte del motor de reglas Jess.

Con 2 Reglas: Axiomas inferidos: 7.

Herramienta	**Protégé-OWL API (mseg)**
OWL + SWRL→ Jess	680
RunJess	49
Jess → OWL	1
Total	730

Tabla 25. Tabla de resultado de las prueba de inferencia #2 por parte del motor de reglas Jess.

Con 3 Reglas: Axiomas inferidos: 12.

Herramienta	Protégé-OWL API (mseg)
OWL+SWRL→ Jess	752
RunJess	61
Jess → OWL	1
Total	814

Tabla 26. Tabla de resultado de las prueba de inferencia #3 por parte del motor de reglas Jess.

Con 4 Reglas: Axiomas inferidos: 16.

Herramienta	Protégé-OWL API (mseg)
OWL+SWRL→ Jess	804
RunJess	63
Jess → OWL	2
Total	869

Tabla 27. Tabla de resultado de las prueba de inferencia #4 por parte del motor de reglas Jess.

De la anterior evaluación fue posible establecer que Protégé-OWL al proveer amplio soporte al motor Jess para la inferencia de reglas, garantiza que la inferencia se realice en un tiempo muy corto y se infieran el total de axiomas que se observan en el Editor Protégé 3.4.7.

- **Resultados y conclusiones:**

Motor de reglas	**Método o algoritmo de razonamiento**	**Sintaxis de lenguaje Reglas adoptado**	**Inclusión de reglas en código**	**Inferencia de axiomas a partir de reglas establecidas**
De Pellet	Acoplamiento del Log de datos del razonador con la aplicación de Reglas-DL	SWRL	Si	Incompleta y con errores
De Jena	Basado en reglas: Puede inferir hechos utilizando encadenamiento hacia adelante, hacia atrás e híbrido	Propio de Jena: reglas definidas por una lista de antecedentes, una lista de consecuentes, un nombre para la regla y una dirección	No	No se visualiza

		que representa la forma en que se resuelve la regla.		
Jess	Algoritmo RETE (Algoritmo de Redundancia Temporal)	SWRL	Si	Completa

Tabla 28. Tabla de resultados generales de las pruebas de inferencia por parte de los motores de reglas de Pellet y Jena.

Los anteriores resultados señalan que si se desea trabajar con el API Jena es preferible utilizar el motor de reglas de Pellet, ya que se tiene mayor control en las reglas, mejor visualización de los resultados en la inferencia y menor tiempo de procesamiento. Tomando como punto de referencia el total de axiomas inferidos para las reglas establecidas en el Editor Protégé 3.4.7, el API de Protégé-OWL, que cuenta con amplio soporte del Sistema Experto de Reglas Jess y toda su funcionalidad, muestra un adecuado soporte de reglas, brindando clases y métodos, que permiten tener control y clara visualización de las reglas que permiten inferir nuevo conocimiento.

3.4.3.3 Evaluación de los lenguajes de consulta sobre los motores más comunes.

Los motores de código abierto más comunes que procesan los lenguajes de consultas sometidos a la evaluación fueron: SPARQL de Jena, SPARQL de Pellet y SQWRL. Se realizaron 5 consultas, las cuales fueron traducidas a los diferentes lenguajes empleados:

	Consulta
Q1	Seleccionar el Estilo, Técnica Artística, Periodo, Tema Artístico, Material y Autor de las Obras de Arte Arquitectónicas
Q2	Seleccionar el Escritor, Género Literario, Estilo, Periodo y Tema Artístico de las Obras Literarias.
Q3	Seleccionar el Periodo, Tema Artístico, Director de Cine, Productor Cinematográfico y Actor de las Obras de Arte Cinematográficas.
Q4	Seleccionar el Estilo, Técnica Artística, Periodo, Tema Artístico, Material y Autor de las Obras de Arte de Pintura.
Q5	Seleccionar el Periodo, Tema Artístico y Estilo de los Objetos de Arte Reliquias.

Tabla 29. Conjunto de consultas para su evaluación.

Para la realización de pruebas de rendimiento, se desarrollaron tres prototipos en Java, con las 5 consultas escritas en lenguaje SPARQL y procesadas por el motor de Jena y de Pellet, mientras que las consultas escritas en lenguaje SQWRL fueron procesadas por el API de consultas SQWRL de Protégé OWL API, tomando las instancias obtenidas como variable de rendimiento. Se probó con la misma ontología: "Arte.owl", con un peso de 80 KB.

Debido a que el motor Fact++ no soporta consultas, no fue posible realizar la evaluación; y en el caso de RacerPro, aunque tiene su propio lenguaje de consulta (nRQL), no ofrece una API en Java para programar las consultas en código, lo cual constituye el principal limitante para su evaluación. Por otro lado, el lenguaje RDQL no fue sometido a evaluación, ya que SPARQL es más extendido y más expresivo.

	Motor de consultas SPARQL de Jena	**Motor de consultas SPARQL de Pellet**	**Motor de consultas SQWRL en ProtégéOWL**
Obra Arquitectónica	BasilicaSanApolinar	BasilicaSanApolinar	BasilicaSanApolinar
Estilo	arquitecturaBizantina	arquitecturaBizantina	arquitecturaBizantina
Técnica Artística	esculpido	esculpido	esculpido
Periodo	altaEdadMedia	altaEdadMedia	altaEdadMedia
Tema Artístico	sagrado	sagrado	sagrado
Material	marmol	marmol	marmol
Autor	teodorico	teodorico	teodorico

Tabla 30. Instancias inferidas en la consulta Q1.

	Motor de consultas SPARQL de Jena	**Motor de consulta SPARQL de Pellet**	**Motor de consulta SQWRL en Protégé OWL**
Obra Literaria	CienAnosdeSoledad	CienAnosdeSoledad	CienAnosdeSoledad
Escritor	GabrielGarciaMarquez	GabrielGarciaMarquez	GabrielGarciaMarquez
Género Literario	novela	novela	novela epico
Estilo	Realismo	realismo	realismo
Periodo	sigloXX	sigloXX	sigloXX
Tema Artístico	Critico	critico	critico

Tabla 31. Instancias inferidas en la consulta Q2.

	Motor de consultas SPARQL de Jena	**Motor de consulta SPARQL de Pellet**	**Motor de consulta SQWRL en Protégé OWL**
Obra Cinematográfica	LaPasionDeCristo	LaPasionDeCristo	LaPasionDeCristo
Periodo	sigloXXI	sigloXXI	sigloXXI
Tema Artístico	Religioso	religioso	religioso
Director de Cine	MelGibson	MelGibson	MelGibson
Productor Cinematográfico	BruceDavey	BruceDavey	BruceDavey
Actor	JamesCaviezel	JamesCaviezel	JamesCaviezel

Tabla 32. Instancias inferidas en la consulta Q3.

	Motor de consultas SPARQL de Jena	**Motor de consultas SPARQL de Pellet**	**Motor de consultas SQWRL en Protégé OWL**
Obra de Pintura	LaMonaLisa	LaMonaLisa	LaMonaLisa
Estilo	Realismo	Realismo	realismo
Técnica Artística	Oleo	Oleo	oleo
Periodo	edadModerna	edadModerna	edadModerna
Tema Artístico	Psicológico	psicologico	psicologico

Material	Barniz	Barniz	barniz
Autor	LeonardoDaVinci	LeonardoDaVinci	LeonardoDaVinci

Tabla 33. Instancias inferidas en la consulta Q4.

	Motor de consultas SPARQL de Jena	Motor de consultas SPARQL de Pellet	Motor de consultas SQWRL en Protégé OWL
Reliquia	LanzaSagrada	LanzaSagrada	LanzaSagrada
Estilo	arteClasico	arteClasico	arteClasico
Periodo	periodoClasico	periodoClasico	periodoClasico
Tema Artístico	Religioso	Religioso	religioso

Tabla 34. Instancias inferidas en la consulta Q5.

- **Resultados y Conclusiones:**

Lenguaje	**Definición**	**Estándar al que es dirigido**	**Lenguaje de origen**	**Herramientas que lo soportan**
SPARQL	RDF Data Access Working Group (DAWG) del Word Wide Web Consortium (W3C)	RDF	RDQL	ARQ un procesador de SPARQL para Jena. Motor de consulta de Pellet. Motor de consulta de KAON2. Protégé: Incluido en el Panel de Consulta SPARQL en Protégé 3.4.x
SQWRL	Semantic Query-enhanced Web Rule Language	OWL	SWRL	Protégé: incluido en SWRLTab Plugin. Protégé OWL API: interfazjava SQWRL.
RDQL	RDF Data Query Language	RDF	SquishQL RDF query Language	Jena API
nRQL	new Racer Query Language	RDF OWL	RQL	Motor de Razonamiento RacerPro

Tabla 35. Resultados de Evaluación de los lenguajes de consulta de ontologías sobre los motores más comunes.

Lenguaje	Total de instancias Inferidas en consultas		
	Motor de consultas SPARQL de Jena	Motor de consultas SPARQL de Pellet	Motor de consultas SQWRL en Protégé OWL
SPARQL	30 instancias correctas.	30 instancias correctas.	No aplica
SQWRL	No aplica.	No aplica.	31 instancias correctas.
RDQL	No se experimentó.	No se experimentó	No se experimentó
nRQL	No se experimentó.	No se experimentó	No se experimentó

Tabla 36. Resultados del total de instancias inferidas en todas las consultas por cada lenguaje sobre los motores más comunes.

Los anteriores resultados señalan que tanto el motor de consulta SPARQL de Jena y Pellet como SQWRL, obtuvieron resultados equivalentes cuando se ejecutaron las consultas. No obstante, es importante resaltar algunas diferencias:

El motor de consultas de Jena está basado en triples RDF, lo cual produce asignaciones de variables que funcionan con un triple a la vez; por lo tanto, cuando no encuentra una instancia la consulta falla, mientras que el motor de consulta de Pellet considera la consulta conjuntiva como un todo y la consulta tiene éxito.

SPARQL es un lenguaje de consultas para RDF, aunque puede ser utilizado para consultar OWL, ya que las ontologías de este tipo pueden ser serializadas a RDF. Sin embargo, este lenguaje no tiene un soporte nativo de OWL ya que su enfoque es sintáctico, a diferencia de SQWRL, el cual es un lenguaje de consultas OWL. Gracias a la eficiencia que proporcionan las reglas, se considera el lenguaje más adecuado, ya que las consultas pueden operar en conjunto con las reglas SWRL y pueden ser usadas para recuperar información inferida por estas reglas con un enfoque semántico. Adicionalmente, SQWRL permite procesar la información obtenida para ser mapeada a datos u objetos en código Java.

CAPÍTULO 4

PROCESAMIENTO DE LA GRAMÁTICA DE CONTEXTO USANDO ONTOLOGÍAS DE DOMINIO

La construcción de un mecanismo que permita el procesamiento de la gramática de contexto de una aplicación móvil Android basada en voz, parte del estudio de los motores ASR de libre distribución, junto con sus capacidades de reconocimiento de voz y configuración, el estudio de los mecanismos de razonamiento ontológico que se pueden aplicar y la definición e integración de los módulos involucrados en dicho procesamiento; en la actualidad, no se encuentran soluciones realmente integradas bajo esta aproximación, específicamente para propósitos de reconocimiento de voz en Android.

De acuerdo al estudio realizado, los dos grandes módulos que integran el mecanismo de procesamiento de la gramática de contexto son: los motores ASR de libre distribución y los componentes de razonamiento ontológico. Puntualmente para el segundo módulo, se combinaron algunos componentes que prevalecían en los trabajos relacionados con dicho razonamiento [113, 116, 134-136], resultando en la adaptación un mecanismo a partir de los existentes. En este sentido, los componentes definidos específicamente para este propósito son: un razonador, un lenguaje de reglas, un lenguaje de consulta (y sus motores de ejecución respectivos) y una API de programación de ontologías.

En este orden de ideas, el capítulo 4 inicia con una breve descripción de los argumentos de la selección de los motores que conformarán dicho mecanismo, los cuales están basados en los resultados y conclusiones obtenidos en el capítulo 3; a continuación, se define la gramática de contexto representada por la ontología de dominio elegida, y una arquitectura base que integra los módulos del mecanismo de procesamiento de dicha gramática de contexto. Finalmente, se presenta un piloto de evaluación, incluyendo los resultados de experimentación y algunas conclusiones relevantes para el estudio.

4.1 SELECCIÓN DE LOS MOTORES ASR Y DE RAZONAMIENTO ONTOLÓGICO.

Para los propósitos del trabajo de grado se eligió el motor de reconocimiento de voz PocketSphinx, dado el soporte de gramática que éste presenta (criterio primordial para su selección) y el rendimiento superior demostrado con respecto a los otros motores sometidos a evaluación. Por su parte, el API de reconocimiento de Android se descartó, ya que fue imposible configurar una gramática propia dentro del mismo, debido a que tal configuración está predefinida en las bases de datos de los servidores de Google, lo cual no ofrece ninguna alternativa de personalización. Finalmente, Julius fue descartado, ya que a pesar de que ofrece la capacidad para configurar la gramática de contexto, no tiene soporte para el idioma español, en el cual se encuentra definida la gramática de contexto.

En cuanto a los motores de razonamiento ontológico, se eligió a Pellet como razonador de ontologías, ya que de acuerdo a la experimentación realizada es el más eficaz y rápido en el proceso de validación de la consistencia y cumplimiento, clasificación y precisiones sobre los conceptos de la jerarquía de la ontología elegida.

Se eligió SWRL como lenguaje de reglas, ya que permite agregar reglas en código y llevar acabo su ejecución por medio del motor especializado Jess. Por consiguiente, se eligió el lenguaje de consultas SQWRL, debido a su especialización en consultar ontologías OWL, y su operación en conjunto con las reglas SWRL, permitiendo recuperar información inferida por éstas reglas.

Para la escogencia del API de Programación, se eligió aquella que soportara los anteriores lenguajes de reglas y consultas, y como se observa en los resultados y conclusiones de la experimentación presentada en el capítulo 3, la única API que soporta dichas características es Protégé OWL API; así mismo, siendo un API de alto nivel, utiliza e incorpora varias funcionalidades de Jena, como por ejemplo cargar modelos en memoria, entre otras.

4.2 DEFINICIÓN DE LA GRAMÁTICA DE CONTEXTO DE UNA APLICACIÓN MÓVIL BASADA EN VOZ REPRESENTADA POR LA ONTOLOGÍA DE DOMINIO ELEGIDA.

En una aplicación móvil basada en voz, la restricción del lenguaje o vocabulario específico a ser utilizado en un dominio, es decir, que pueda ser reconocido por un motor ASR, se define como la gramática de contexto y puede ser representada por medio de ontologías de dominio. En ese sentido, dicho lenguaje de entrada de voz es una instanciación de la ontología de dominio y está destinado a tener acceso a un vocabulario de una gramática predefinida almacenado. Adicionalmente, ésta gramática para el reconocedor de voz elegido es escrita manualmente en el formato permitido por el ASR, en este caso PocketSphinx emplea el formato JSGF (JSpeechGrammarFormat) [58].

De acuerdo a lo anterior, la gramática escrita en JSGF se compone de un conjunto de reglas que definen las palabras que se pueden pronunciar para ser reconocidas. Estas reglas son combinaciones de texto decible y referencias a otras, las cuales se representan por su nombre entre los caracteres “<>”. De igual forma, la gramática también está compuesta por “tokens”, los cuales son una secuencia de caracteres delimitados por espacios en blanco, entre comillas, o delimitada por otros símbolos que son importantes en la gramática, por ejemplo = | * + <> () [] {} / * * / / /.

En este caso, un token es una referencia a una entrada del vocabulario o léxico del reconocedor, el cual define la pronunciación del “token”, y con esta pronunciación el reconocedor es capaz de escucharlo.

Como se mencionó anteriormente, la gramática es una instanciación de la ontología, es decir el mapeo de la ontología a las reglas de gramática. Para el caso específico del trabajo de grado, se emplean los conceptos e instancias de la ontología como los “tokens” a ser escuchados por el reconocedor, y las relaciones entre las clases se mapean a las combinaciones lógicas C*ompositions* y *Grouping*, explicadas en el capítulo 2, las cuales se emplean según las distintas posibilidades de pronunciación que se deseen programar para las consultas que son equivalentes. Dentro de las anteriores combinaciones lógicas, se emplearon explícitamente: Secuencias, Alternativas, Parentesis y Agrupación Opcional, también descritas en el capítulo 2, para la escritura de la gramática de contexto basada en la ontología de dominio “Arte.owl”.

4.3 ARQUITECTURA BASE PARA EL PROCESAMIENTO DE LA GRAMÁTICA DE CONTEXTO USANDO ONTOLOGÍAS DE DOMINIO.

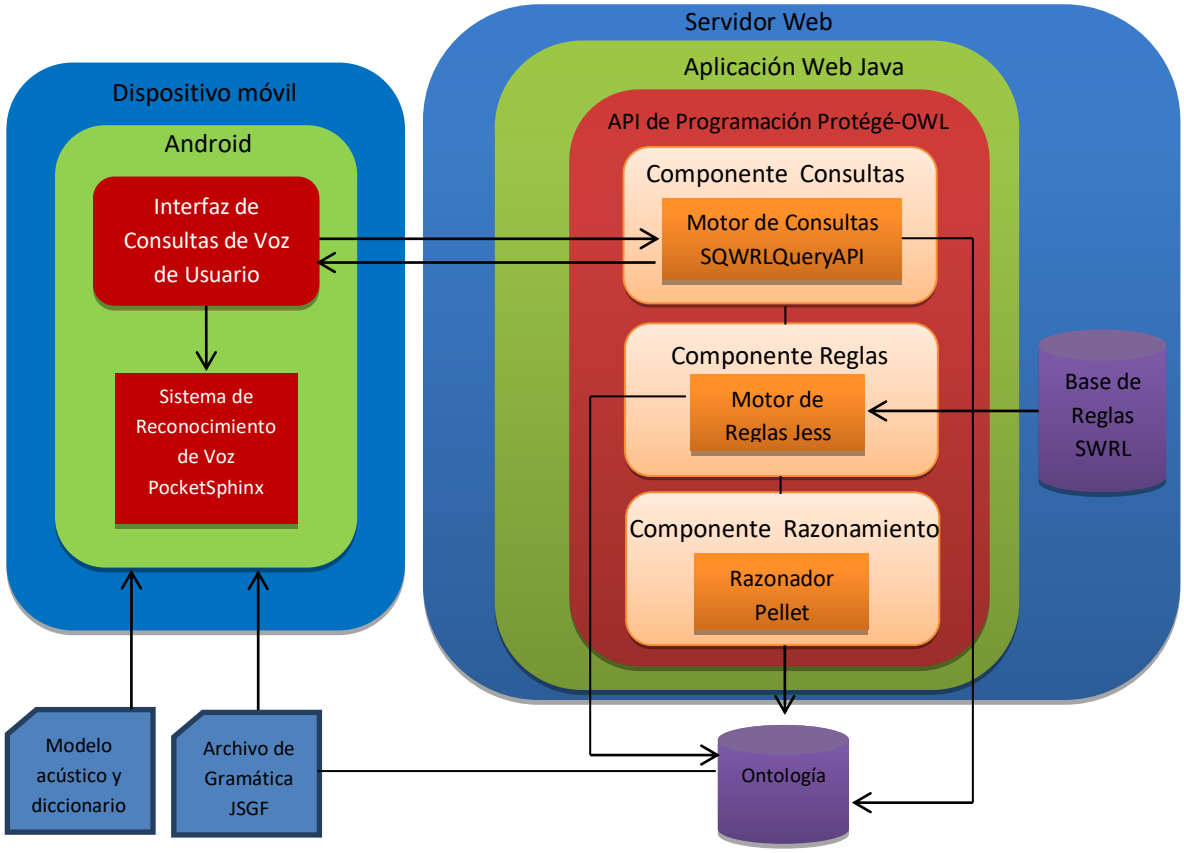

Figura 10. Arquitectura base para el procesamiento de la gramática de contexto.

La arquitectura base para el procesamiento ontológico de la gramática de contexto de una aplicación móvil basada en voz para clientes Android usando ontologías de dominio, presenta un arquitectura cliente-servidor; a nivel de implementación, se toma como referencia un cliente móvil Android, y una aplicación Web Java que se ejecuta sobre Apache Tomcat, como módulo servidor.

La elección de una arquitectura tipo cliente-servidor obedece a que no sólo es un referente en los entornos de telefonía móvil dadas las características de interacción (la mayor parte del procesamiento recae en el servidor para minimizar los requerimientos computacionales del cliente)[146], sino a las características propias que requiere el procesamiento de la gramática de contexto. Los cálculos de razonamiento complejo, en los que se derivan nuevo conocimiento a partir de una ontología existente, sólo se pueden realizar en el lado del servidor, ya que requiere tiempo y alta capacidad de procesamiento, características que pese a su evolución hardware, son incipientes en los teléfonos móviles actuales para el propósito planteado.

A continuación se describen cada uno de los componentes de la arquitectura:

4.2.1 Cliente.

El cliente está compuesto de una interfaz de voz, encargada de la interacción con el usuario por medio de consultas al sistema, y un motor de reconocimiento autómatico del habla embebido en el dispositivo: PocketSphinx, el cual obtiene la palabra o frase pronunciada por el usuario.

En este sentido, para realizar el reconcimiento de voz, PocketsSphinx emplea un modelo acústico para la lengua española, un diccionario de pronunciación y un archivo de gramática escrita en formato JSGF, la cual representa el "vocabulario de entrada" restringido, y es una instanciación de la ontología de dominio "Arte.owl" escogida. Dicha gramática ofrece la posibilidad del reconocimiento de las frases que son pronunciadas dentro del contexto especificado de una manera diferente y obtienen la misma información para una búsqueda específica.

4.2.2 Servidor.

El servidor está implementado a través de Java Servlets y emplea el protocolo de comunicación HTTP para enviar la respuesta de nuevo al dispositivo móvil de las frases reconocidas que han sido enviadas por el cliente a través de peticiones con el método GET.

4.2.3 API de programación Protégé-OWL.

Protégé OWL API se emplea como API de programación de ontologías y se encarga de cargar el modelo ontológico en memoria y acceder a un razonador externo para realizar inferencias acerca de las clases y las instancias en la ontología, por medio de la interfaz DIG. La ingerfaz DIG es un estándar que proporciona una especificación para la conexión con razonadores DL externos a través de HTTP, usando el lenguaje DIG, el cual es una representación basada en XML de las entidades ontológicas, como clases, propiedades, instancias y axiomas. Acutalmente DIG sólo es soportada por las distribuciones de Protégé 3.4.x [147].

4.2.4 Motor de consultas SQWRLQueryAPI

El componente de consultas se encarga de recibir la frase reconocida por la aplicación móvil y realizar la respectiva consulta a la ontología empleando el motor SQWRLQueryAPI, el cual es un subsistema de SWRLTab que proporciona una interfaz similar a JDBC en Java para recuperar el resultado de las consultas SQWRL. Esta API está disponible en la distribución Protégé-OWL 3.4.x.

Para la creación de un motor de consulta SQWRL y ejecución de las mismas en una ontología, utiliza la interfaz *SQWRLQueryEngine*, por medio de la clase *SQWRLQueryEngineFactory*.

La ejecución de consultas que utilizan esta API no modifica la ontología OWL subyacente, es decir, los resultados se mantienen en el interior del motor de consulta únicamente. Como se ha mencionado anteriormente, las consultas SQWRL se pueden ejecutar junto con las reglas SWRL, y por lo tanto, pueden acceder a este conocimiento inferido [148].

4.2.5 Motor de reglas Jess

Las reglas se han incluido a la ontología por medio de la interfaz *SWRLRuleEngineBridge*, el cual es un subcomponente del SWRLTab que proporciona la infraestructura necesaria para incorporar motores de reglas en Protégé-OWL permitiendo su ejecución. Dicha interfaz proporciona mecanismos para:

- Importar reglas SWRL y clases OWL relevantes, instancias y propiedades de un modelo OWL y escribir el conocimiento inferido de un motor de reglas.
- Permitir que el motor de reglas lleve a cabo la inferencia y hacer valer su nuevo conocimiento hacia el puente.
- Insertar el conocimiento en un modelo OWL.

Por otra parte, la interfaz *TargetSWRLRuleEngine* define los métodos que un motor de reglas debe implementar; dichos métodos se utilizan para representar reglas SWRL y clases OWL, propiedades, instancias, y axiomas dentro de un motor de reglas en particular; de igual forma, se proporciona un método para realizar inferencia utilizando este conocimiento.

Específicamente, para el motor de reglas Jess se suministra una implementación, con la distribución estándar de Protégé-OWL, llamada *SWRLJessTab*. Dicha implementación requiere que el motor de reglas Jess se encuentre instalado en el sistema y se debe descargar separadamente de su página oficial. Este motor también es necesario para la ejecución de las consultas SQWRL mencionadas en el ítem anterior.

4.2.6 Razonador Pellet

El razonador Pellet es accedido a través de la API de razonamiento de Protégé-OWL, la cual se resume en el paquete *edu.stanford.smi.protegex.owl.inference*, y las principales clases que se utilizan son: *ReasonerManager* (utilizado para obtener un razonador) y *ProtegeOWLReasoner* (una interfaz para el razonador externo DIG).

En general, el primer paso cuando se utiliza la API de razonamiento es la obtención de una instancia de *ProtegeOWLReasoner* de un modelo OWL. Esta instancia del razonador se puede utilizar para obtener información sobre el modelo inferido: superclases, clases equivalentes, y tipos de instancias. La comunicación externa con un razonador DIG compatible se facilita a través de HTTP, por lo que se proporciona la dirección URL del razonador, y en seguida se procede a probar la conexión del mismo y a clasificar la ontología [147].

Para el caso específico de Pellet, éste razonador se debe iniciar a través de su interfaz DIG y el número de puerto sobre el que se encuentre corriendo, debe ser el mismo que se especifique en la URL en Protégé-OWL API.

4.2.7 Interacción entre los módulos

A continuación se demuestra la interacción entre los módulos y componentes para realizar todo el proceso de reconocimiento de voz y razonamiento ontológico por medio de diagramas de secuencia.

La primera interacción que ocurre es entre el usuario y el cliente, es decir la aplicación móvil Android, que cuenta con una interfaz de consultas de voz que se comunica con el motor ASR embebido PocketSphinx, el cual, en primer lugar toma las secuencia de fonemas pronunciados por el usuario como entrada y, combinado el modelo acústico y el diccionario de pronunciación, encuentra las palabras más probables para esta secuencia, para posteriormente pasarlas a la gramática JSGF configurada, y realizar su comparación con las frases programadas, lo que finalmente retorna la secuencia "optima" de palabras, es decir, la que mejor se ajuste a los vectores de las características espectrales de la señal de entrada.

La aplicación está programada para reconocer exclusivamente las sentencias programadas en la gramática, la cual contiene todas las palabras y expresiones que pueden ser utilizadas para un contexto determinado, en este caso, consultas de Obras u Objetos de arte del Museo de Arte Religioso. En consecuencia, una palabra o expresión que no es parte de las sentencias que se definen, no es reconocida por la aplicación.

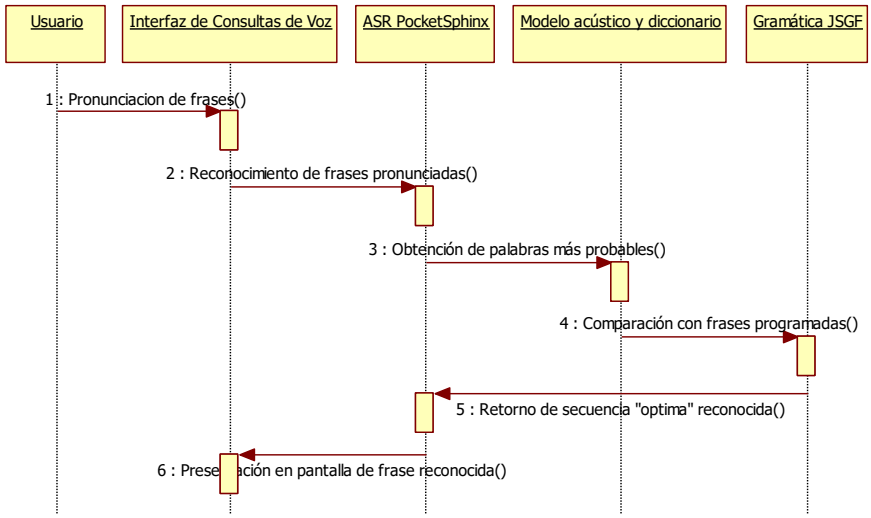

Figura 11. Diagrama de secuencia Interacción entre el Usuario y el dispositivo cliente

En segundo lugar, al iniciar la aplicación Web Java ejecutada sobre el servidor Apache Tomcat, se razona la ontología empleando el motor Pellet por medio de la interfaz DIG de Protégé sobre HTTP, el cual ha sido previamente arrancado sobre el localhost y el puerto 8081, y se encarga de proporcionar sus servicios de inferencia, es decir validación de consistencia y clasificación de la ontología. En seguida, se toma la base de reglas creadas y se aplican a la ontología por medio de la clase *SWRLRuleEngineBridge*, para que el motor de reglas empleado, en éste caso Jess, las ejecute e infiera conocimiento a partir de ellas. Dicho conocimiento inferido podrá ser consultado posteriormente.

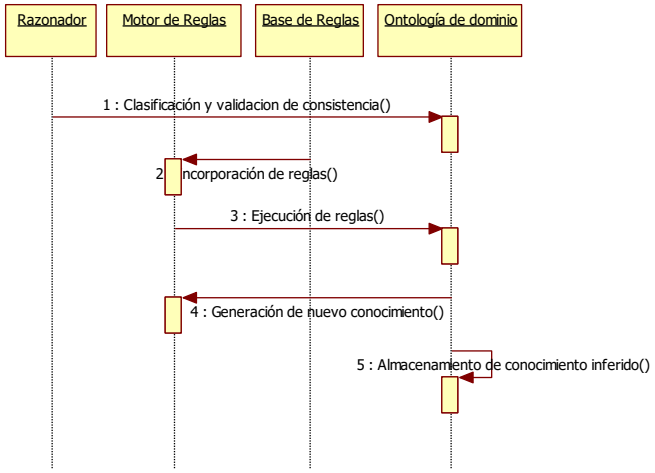

Figura 12. Diagrama de secuencia Interacción entre razonador y motor de reglas

Finalmente, el servidor usa Protégé OWL API como el API de programación de ontologías y se comunica con la aplicación móvil por medio del protocolo HTTP. Se encarga de recibir la frase reconocida por la aplicación móvil y realizar la respectiva consulta a la ontología y al nuevo conocimiento inferido adicionado anteriormente, para enviar la respuesta de nuevo al dispositivo móvil, empleando el motor de consultas de SQWRL, disponible en la distribución de Protégé, por medio de la SQWRLQueryAPI.

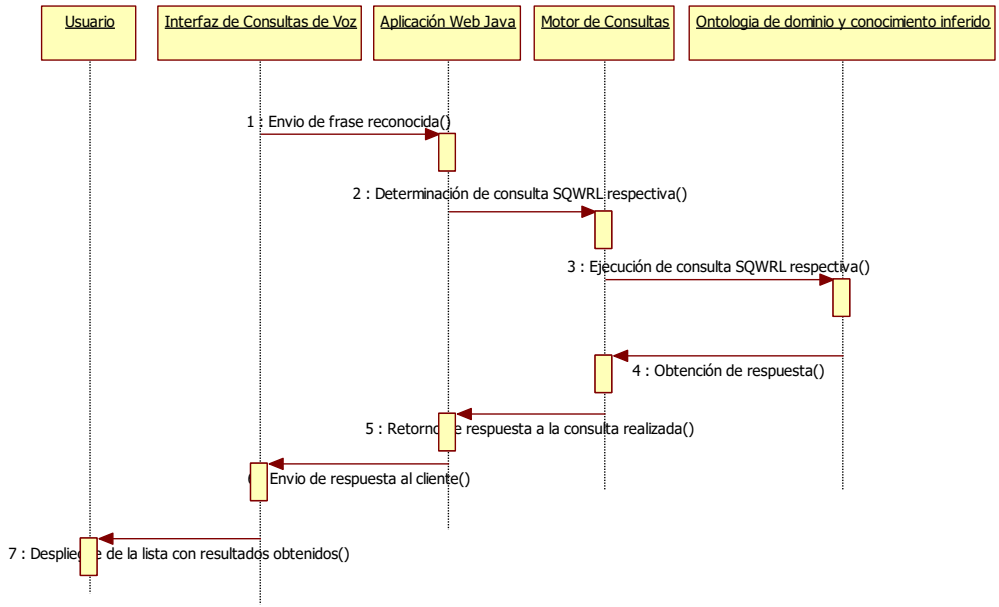

Figura 13. Diagrama de secuencia Interacción entre cliente y servidor

Finalmente la aplicación Web Java, envía al dispositivo móvil, la lista de resultados de la consulta a la ontología por medio de HTTP, el cual se encarga de desplegarla al usuario, para obtener información detallada de la misma.

4.3 PILOTO: CLIENTE MÓVIL CON PROCESAMIENTO DE LA GRAMÁTICA DE CONTEXTO USANDO ONTOLOGÍAS DE DOMINIO

4.3.1 Descripción del piloto

Para la evaluación del mecanismo de procesamiento definido, se construyó un Piloto de una aplicación móvil con interacción a través de voz, la cual se lleva a cabo por medio de consultas al sistema y las respuestas son desplegadas en la pantalla del dispositivo móvil. Las consultas que se podrán realizar están relacionadas con las Obras u Objetos de Arte del Museo de Arte Religioso de Popayán.

El objetivo del piloto es evaluar las capacidades del mecanismo previamente definido para procesar adecuadamente la gramática de contexto definida para un dominio específico de aplicación y retornar información consistente.

Para la interacción por medio de la voz se empleó el motor de reconocimiento de voz PocketSphinx, y para procesar la consulta y obtener respuesta a la misma, se empleó el mecanismo de razonamiento ontológico previamente descrito, integrado por el razonador Pellet, el motor de reglas Jess, el motor de consultas, es decir SQWRLQueryAPI y finalmente el API de programación Protégé-OWL.

A continuación, se presentan el modelo de casos de uso de la aplicación y su respectiva descripción.

4.3.2 Modelo de casos de uso de la aplicación

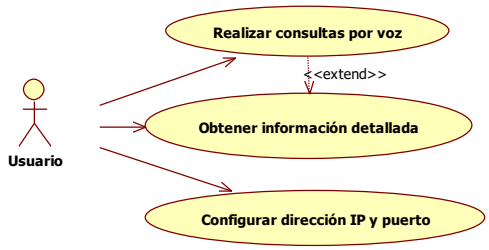

Figura 14. Modelo de casos de uso de la aplicación del Piloto

- **Descripción de casos de uso de alto nivel**

Caso de uso	Configurar Dirección IP y puerto
Objetivo	Fijar la dirección IP y el puerto al que va a conectarse la aplicación móvil y sobre los cuales se encuentra corriendo el Servidor Web.
Actor /es	Usuario
Descripción	Este caso de uso inicia cuando el usuario selecciona la opción “Configurar IP” del menú principal, en seguida se presentará una interfaz con dos campos editables de texto, en los cuales el usuario debe ingresar la dirección IP y el número de puerto del servidor Web.

Caso de uso	Realizar consultas por voz
Objetivo	Realizar consultas a la aplicación, relacionadas con las Obras de Arte u

	Objetos de Arte, consultando todas las existentes por un Nombre, Periodo, Autor o Técnica Artística específicos.
Actor /es	Usuario
Descripción	Este caso de uso inicia después de que el usuario selecciona la opción "Consultar por voz" del menú principal, en seguida se presentará la interfaz que permite realizar la consulta a la aplicación, y por medio del botón "Presione para hablar" se realiza el proceso de reconocimiento, mostrando en pantalla la frase que finalmente fue reconocida en el cuadro de texto de dicha interfaz.

Caso de uso	Obtener información detallada
Objetivo	Acceder a la información detallada de una Obra u Objeto de arte seleccionado, que se encuentre dentro de la lista respectiva que trae consigo el servidor como respuesta a las consultas que han sido emitidas desde el cliente.
Actor /es	Usuario
Descripción	Este caso de uso inicia después de que el usuario selecciona una opción de la lista obtenida en el anterior caso de uso, para obtener mayor información, la cual será desplegada en pantalla mostrando detalles más específicos de las Obras u Objetos de arte consultados como: Título, Autor, Tecnica, Material, Tema y Estilo.

4.4 IMPLEMENTACIÓN DE LA ARQUITECTURA BASE PARA EL PROCESAMIENTO DE LA GRAMÁTICA DE CONTEXTO.

Para la construcción del piloto del trabajo de grado, se siguió la arquitectura base tipo cliente-servidor para el procesamiento ontológico de la gramática de contexto de una aplicación móvil basada en voz para clientes Android usando ontologías de dominio, presentada en la sección X.X.

En el lado del cliente, se encuentra una aplicación móvil Android que se encarga de realizar el reconocimiento automático del habla, empleando el motor embebido PocketSphinx, el cual emplea el Modelo Acústico para la Lengua Española disponible en la página de CMU Sphinx [131], el diccionario de pronunciación del proyecto "The Automatic Speech Recognition for Spanish Language Project" [149], el cual fue modificado para incluir todas las palabras de la gramática y su respectiva pronunciación, y finalmente, un archivo de gramática JSGF escrito manualmente basado en la ontología de domino "Arte.owl", el cual representa las posibles frases de consulta de voz que el usuario podrá realizar a la aplicación. Para la interacción con el usuario, la aplicación consta de una interfaz de consultas de voz de usuario con un estilo representativo de un museo religioso. Para la comunicación con el servidor, el dispositivo móvil emplea el protocolo HTTP, a través del método GET, enviando la frase previamente reconocida como un parámetro en la URL.

Adicionalmente, se creó una base de datos dentro de la aplicación móvil, por medio del motor de bases de datos SQLite [150], la cual contiene una tabla con las siguientes columnas: Título, Autor, Periodo, Tecnica, Material, Tema y Estilo, para almacenar en ella, la lista de Obras u Objetos de arte respectiva que trae consigo el servidor como respuesta a una consulta realizada.

Por otra parte, a la ontología elegida "Arte.owl", se le incluyeron 40 instancias como Obras de Arte y 12 como Objetos de Arte, a cada uno de los cuales se le fijaron manualmente las propiedades de Nombre, Material, Género, Periodo, Autor y Género. La propiedad que no se fijó manualmente fue la de Estilo, ya que el establecimiento de reglas que relacionen el Periodo con ésta última, permite la inferencia del Estilo de cada instancia, de acuerdo al Periodo que tenga relacionado, proporcionando nuevo conocimiento que no se encuentra explícitamente en la ontología.

En el lado del servidor, éste se encuentra implementado a través de Java Servlets y Protégé OWL API como el API de programación de ontologías. Una vez se inicia la aplicación Java Web sobre el servidor Apache Tomcat, corriendo sobre el localhost y el puerto 9416, se conecta con el razonador Pellet por medio de la interfaz DIG de Protégé sobre HTTP, el cual previamente ha arrancado sobre el localhost y el puerto 8081, para razonar la ontología y proporcionar sus servicios de inferencia, es decir validación de consistencia y clasificación. Seguidamente se le incluyen a la ontología las reglas por medio de la clase *SWRLRuleEngineBridge*, subcomponente de *SWRLTab* proporcionada por Protégé, encargada de proporcionar un puente entre la ontología con las reglas SWRL incluidas y el motor de reglas Jess, el cual ejecuta las reglas e infiere conocimiento a partir de ellas, dicha inferencia puede ser consultada en cualquier momento.

Una vez llega una solicitud HTTP al servidor, éste recibe la frase enviada y realiza la respectiva consulta a la ontología para enviar la respuesta de nuevo al dispositivo móvil, empleando el motor de consultas de SQWRL, disponible en la distribución de Protégé, por medio de la SQWRLQueryAPI.

La configuración del entorno de prueba, incluyó un teléfono móvil con sistema operativo Android 2.3 Samsung Google Nexus S con 16 GB de memoria interna, 512 MB RAM y procesador ARM Cortex A8 1GHz. El tipo de conexión HTTP empleada entre el dispositivo y el servidor es inalámbrica Wi-Fi con un ancho de banda disponible de 11Mbps. Para ejecutar el servidor se empleó una máquina con las siguientes características: 320 GB de disco duro, 4GB RAM, Procesador Intel Core 2Duo 2.2GHz y tarjeta inalámbrica 802.11b. A continuación se muestra la interacción entre los componentes del entorno de pruebas empleado:

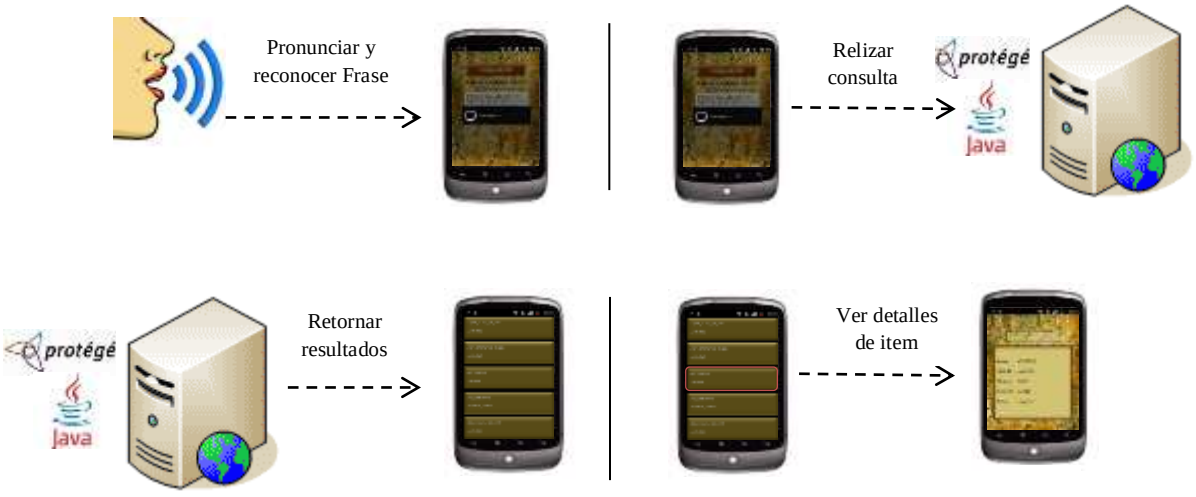

Figura 15. Diagrama de interacción entre el usuario, el dispositivo móvil y el servidor Web

4.5 DESCRIPCIÓN DEL PILOTO

Para la construcción del piloto básicamente se programaron varios tipos de consultas, permitiendo que para una búsqueda específica, la variación en la pronunciación de frases, obtuviera una misma información de una manera consistente. Adicionalmente, se incluyeron en código 4 reglas que relacionaran el Periodo de una Obra u Objeto de arte con un Estilo respectivo, para inferir específicamente el Estilo al que pertenece cada una de ellas. A continuación, se presenta la navegabilidad de la aplicación por medio de las interfaces que conforman el piloto, y una breve explicación de cada una de ellas.

4.5.1 Interfaces Básicas de la Aplicación

Interfaz inicial — Menú Principal — Configurar IP

Consultar por Voz — Procesando consulta — Lista de resultados

Detalles Selección

Figura 16. Interfaces y mapa de navegabilidad de la aplicación

La aplicación inicialmente muestra un Splash Android con la imagen más representativa del museo de Arte Religioso de la ciudad de Popayán, "la Virgen del Apocalipsis", obra atribuida al maestro mestizo quiteño Bernardo de Legarda quien vivió entre 1700 y 1773; a continuación se presenta al usuario la interfaz con las opciones del menú principal, donde se podrá seleccionar la opción deseada: configurar la dirección IP y el puerto del servidor al que la aplicación se va a conectar, iniciar el proceso de consultas por voz, consultar la ayuda del sistema o finalmente, salir de la aplicación.
La elección de la opción "Consultar por voz" por parte del usuario presentará la interfaz que permite realizar la consulta a la aplicación, y por medio del botón "Presione para hablar" se realiza el proceso de reconocimiento, mostrando en pantalla la frase que finalmente fue reconocida. La siguiente interfaz muestra la lista con el nombre y autor de las Obras u Objetos de arte, obtenidos como respuesta a la consulta realizada. En este momento, el usuario puede seleccionar una opción de la lista, para obtener mayor información, la cual será desplegada en pantalla mostrando detalles más específicos de las Obras u Objetos de arte consultados.

4.5.2 Posibles Consultas.

Las variaciones de frases a reconocer para cada tipo de consulta realizada a la aplicación son mostradas a continuación. En total son los 7 tipos permitidos.

- Consultar todas las Obras u Objetos de Arte

LISTAR/MOSTRAR TODAS(OS) LAS(OS) OBRAS/OBJETOS DE ARTE

- Consultar las Obras de Arte especificando el nombre

MOSTRAR LA OBRA DE ARTE LLAMADA/CON NOMBRE/DENOMINADA "parametroNombre":

- Posibles nombres obras de arte ("parametronombre"):

ADORACION CINCO RAZAS | ARCANGEL SAN MIGUEL | COLECCION DEL APOSTOLADO | EL BAUTISMO DE JESUS | EL MARTIRIO DE SANTA BARBARA | INMACULADA DEL

APOCALIPSIS | LA PIEDAD | LA SAGRADA FAMILIA Y SAN LORENZO | LA VIRGEN CON EL NIñO | LA VIRGEN DEL SILENCIO | MADRE MARIANA DE SAN ESTANISLAO Y SAA | NACIMIENTO | SANTA ANA Y LA VIRGEN | SANTA CLARA DE ASIS | SANTIAGO EL MAYOR | SANTIAGO EL MENOR | SANTO TOMAS | SAN AGUSTIN | SAN AMBROSIO | SAN ANTONIO DE PADUA | SAN FELIPE | SAN FRANCISCO DE ASIS | SAN GREGORIO MAGNO | SAN JERONIMO | SAN JUAN | SAN JUDAS TADEO | SAN LUCAS | SAN MARCOS | SAN MATEO | SAN MATHIAS | SAN PABLO | SAN SIMON | SAN VICENTE FERRER | SERIE MARIANA | VIRGEN DEL CALVARIO | VIRGEN DEL ROSARIO Y PURGATORIO | VIRGEN DE LA ALMUDENA

- Consultar los Objetos de Arte especificando el nombre

MOSTRAR EL OBJETO DE ARTE LLAMADO/CON NOMBRE/DENOMINADO "parametroNombre":

- Posibles nombres objetos de arte ("parametronombre"):

ANGELES | ARCANGEL SAN GABRIEL | ARCANGELES | CALVARIO CRISTO AGONIZANTE | CRUCIFIJO CRISTO AGONIZANTE | INMACULADA CONCEPCION | INMACULADA DEL APOCALIPSIS | NINO DE LA PASION | NIñO JESUS | RETABLO CON CRISTO | SAN JOAQUIN SANTA ANA Y LA VIRGEN NIñA | VIRGEN DEL TRANSITO

- Buscar Obras/Objetos de Arte por un siglo específico

LISTAR/MOSTRAR LAS(OS)/TODAS(OS) LAS(OS) OBRAS/OBJETOS DE ARTE DEL SIGLO "parametroSiglo"

- Posibles Siglos de obras/objetos de arte ("parametroSiglo"):

DIECISEIS | DIECISIETE | DIECIOCHO | DIECINUEVE | VEINTE | VEINTIUNO

- Buscar Obras/Objetos de Arte por una técnica específica

LISTAR/MOSTRAR LAS(OS) OBRAS/OBJETOS DE ARTE HECHAS(OS) CON "parametroTecnica"
LISTAR/MOSTRAR LAS(OS) OBRAS/OBJETOS DE ARTE QUE ESTÁN HECHAS(OS) CON "parametroTecnica"
LISTAR/MOSTRAR LAS(OS) OBRAS/OBJETOS DE ARTE REALIZADAS(OS) CON "parametroTecnica"
LISTAR/MOSTRAR LAS(OS) OBRAS/OBJETOS DE ARTE QUE ESTÁN REALIZADAS(OS) CON "parametroTecnica"
LISTAR/MOSTRAR LAS(OS) OBRAS/OBJETOS DE ARTE HECHAS(OS) CON LA TÉCNICA "parametroTecnica"
LISTAR/MOSTRAR LAS(OS) OBRAS/OBJETOS DE ARTE QUE ESTÁN HECHAS(OS) CON LA TÉCNICA "parametroTecnica"
LISTAR/MOSTRAR LAS(OS) OBRAS/OBJETOS DE ARTE REALIZADAS(OS) CON LA TÉCNICA "parametroTecnica"

LISTAR/MOSTRAR LAS(OS) OBRAS/OBJETOS DE ARTE QUE ESTÁN REALIZADAS(OS) CON LA TÉCNICA “parametroTecnica”
LISTAR/MOSTRAR LAS(OS) OBRAS/OBJETOS DE ARTE QUE UTILIZAN “parametroTecnica”
LISTAR/MOSTRAR LAS(OS) OBRAS/OBJETOS DE ARTE QUE UTILIZAN LA TÉCNICA “parametroTecnica”
LISTAR/MOSTRAR LAS(OS) OBRAS/OBJETOS DE ARTE QUE EMPLEAN “parametroTecnica”
LISTAR/MOSTRAR LAS(OS) OBRAS/OBJETOS DE ARTE QUE EMPLEAN LA TÉCNICA “parametroTecnica”
LISTAR/MOSTRAR LAS OBRAS DE ARTE PINTADAS CON “parametroTecnica”
LISTAR/MOSTRAR LAS OBRAS DE ARTE PINTADAS CON LA TÉCNICA “parametroTecnica”
LISTAR/MOSTRAR LAS OBRAS DE ARTE QUE ESTÁN PINTADAS CON “parametroTecnica”
LISTAR/MOSTRAR LAS OBRAS DE ARTE QUE ESTÁN PINTADAS CON LA TÉCNICA “parametroTecnica”
LISTAR/MOSTRAR LAS(OS) OBRAS/OBJETOS DE ARTE ELABORADAS(OS) CON “parametroTecnica”
LISTAR/MOSTRAR LAS(OS) OBRAS/OBJETOS DE ARTE QUE ESTÁN ELABORADAS(OS) CON “parametroTecnica”
LISTAR/MOSTRAR LAS(OS) OBRAS/OBJETOS DE ARTE QUE ESTÁN ELABORADAS(OS) CON LA TÉCNICA “parametroTecnica”
LISTAR/MOSTRAR LAS(OS) OBRAS/OBJETOS DE ARTE ELABORADAS(OS) CON LA TÉCNICA “parametroTecnica”
LISTAR/MOSTRAR LAS(OS) OBRAS/OBJETOS DE ARTE QUE USAN “parametroTecnica”
LISTAR/MOSTRAR LAS(OS) OBRAS/OBJETOS DE ARTE QUE USAN LA TÉCNICA “parametroTecnica”
LISTAR/MOSTRAR LAS(OS) OBRAS/OBJETOS DE ARTE QUE MANEJAN “parametroTecnica”
LISTAR/MOSTRAR LAS(OS) OBRAS/OBJETOS DE ARTE QUE MANEJAN LA TÉCNICA “parametroTecnica”

- Posibles Técnicas artísticas de Obras de Arte (“parametroTecnica”):

OLEO SOBRE LIENZO | OLEO SOBRE MARMOL | OLEO SOBRE MADERA | OLEO SOBRE COBRE | GRABADO SOBRE PAPEL

- Posibles Técnicas artísticas de Objetos de Arte (“parametroTecnica”):

TALLADO SOBRE MADERA

- Buscar Obras de Arte por un autor específico

LISTAR/MOSTRAR LAS OBRAS DE ARTE DEL AUTOR/PINTOR/ARTISTA “parametroAutor”

- Posibles autores Obras de Arte (“parametroAutor”):

JOSE CORTEZ | JUAN DE JUANES | MANUEL RAMIREZ | MANUEL SEPULVEDA | PIARRETTA Y PITTERI | RAFAEL TROYA | VICENTE ALBAN | ANONIMO

- Buscar Objetos de Arte por un autor específico

LISTAR/MOSTRAR LOS OBJETOS DE ARTE DEL AUTOR/ARTISTA “parametroAutor”

- Posibles autores Objetos de Arte (parametroAutor):

BERNARDO DE LEGARDA | MANUEL CASPICARA | TALLADOR ANONIMO

4.5 EVALUACIÓN DEL PROCESAMIENTO DE LA GRAMÁTICA DE CONTEXTO

Para efectuar la evaluación, se han definido los siguientes criterios:

No. Criterio	Criterio de evaluación
1	Métricas de referencia en evaluación de ASR
2	Latencia
3	Correcta respuesta a las consultas de voz
4	Inferencia de nuevo conocimiento por parte de las reglas

Tabla 37. Criterios de evaluación para el piloto.

Con respecto al criterio 1, se medirán las métricas de referencia que son usadas más comúnmente en la evaluación de los ASR; el criterio 2 se refiere al tiempo de latencia, es decir el tiempo que se tarda en completar una solicitud de búsqueda por la voz, y más precisamente, el tiempo desde que el usuario termina de hablar hasta que los resultados de la búsqueda aparecen en pantalla [20]. El criterio 3 se refiere a la correcta respuesta a cada una de las consultas de voz realizadas por el usuario, es decir la correspondencia entre el comando de voz y la respuesta desde del servidor. Para finalizar, el último criterio hace referencia a la inferencia de nuevo conocimiento en las instancias creadas, es decir el Estilo de cada instancia partir de las reglas incluidas.

4.5.1 Plan de Pruebas

4.5.1.1 Evaluación de las Métricas de referencia en evaluación de ASR

Para la medición de las métricas WER, la WRR y la WCR, se realizaron una serie de pruebas consistentes de un número de iteraciones que corresponden al número de variaciones de frases para cada tipo de consulta especificadas en el ítem 5.3.2, y el registro del número de inserciones, supresiones y sustituciones por cada una de ellas. Los resultados obtenidos son registrados en la siguiente tabla:

$$WER = \frac{S+D+I}{Nr} \qquad WRR = 1 - WER = \frac{H-I}{Nr} \qquad WCR = \frac{H}{Nr}$$

Tipo de Consulta	**Iteraciones o variaciones de frases**	**Numero Sustituciones (D)**	**Numero Supresiones (S)**	**Numero Inserciones (I)**	**Numero Correctas (H)**	**WER**	**WRR**	**WCR**
Consultar todas las Obras u Objetos de Arte	4	0	0	0	4	0	1	1

Consultar las Obras de Arte especificando el nombre	37	2	0	0	35	0,05	0,95	0,95
Consultar los Objetos de Arte especificando el nombre	11	1	0	0	10	0,09	0,90	0,90
Buscar Obras u Objetos de Arte por un siglo específico	12	0	0	0	12	0	1	1
Buscar Obras u Objetos de Arte por una técnica específica	44	0	0	0	44	0	1	1
Buscar Obras de Arte por un autor específico	24	3	0	0	21	0,125	0,875	0,875
Buscar Objetos de Arte por un autor específico	6	0	0	0	6	0	1	1

Tabla 38. Resultados obtenidos en la medición de métricas de evaluación del reconocimiento de voz

WER Promedio	WRR Promedio	WRC Promedio
0,038	0,96	0,96

Tabla 39. Resultados de la WER, WRR y WRC promedio obtenidos.

Como se puede observar en los anteriores resultados, la tasa correcta de reconocimiento de voz de las frases programadas en la aplicación es alta, demostrando un buen desempeño para el reconocedor elegido. Sin embargo, existieron algunas frases que no se reconocieron correctamente, específicamente las últimas consultas, relacionadas con buscar Obras/Objetos de arte por un autor específico, debido a que en la gramática eran las últimas que se encontraban programadas, así que existió la necesidad de

repetir algunas de ellas. Esto indica que si una gramática se extiende demasiado, genera ambigüedad en el reconocimiento.

4.5.1.2 Evaluación de Latencia

Para la medición de la latencia, se realizaron una serie de pruebas que consistieron en elegir dos variaciones de frases a reconocer para un mismo tipo de consulta y realizar la comparativa entre tres dispositivos móviles Android, con diferentes especificaciones técnicas:

Dispositivo	**Memoria Interna**	**Memoria RAM**	**Procesador**
Samsung Google Nexus S	16 GB	512 MB RAM	Procesador ARM Cortex A8 1GHz
Huawei U8150 IDEOS	512 MB	256 MB RAM	Procesador 533 MHz
Samsung Galaxy 550	170 MB	512 MB RAM	Procesador 600 MHz

Tabla 40. Especificaciones técnicas de dispositivos empleados

Los resultados de la medición de tiempos de latencia en cada dispositivo son registrados en la siguiente tabla:

Tipo de Consulta	**Frase a reconocer**	**Tiempo de latencia**		
		Samsung Nexus S	**Huawei Ideos**	**Samusng Galaxy 550**
Consultar todas las Obras u Objetos de Arte	LISTAR TODAS LAS OBRAS DE ARTE	6 seg	12 seg	11 seg
	LISTAR TODOS LOS OBJETOS DE ARTE	3 seg	9 seg	9 seg
Consultar las Obras de Arte especificando el nombre	MOSTRAR LA OBRA DE ARTE CON NOMBRE ADORACION CINCO RAZAS	2 seg	10 seg	9 seg
	MOSTRAR LA OBRA DE ARTE DENOMINADA COLECCION DEL APOSTOLADO	2 seg	10 seg	10 seg
Consultar los Objetos de Arte especificando el nombre	MOSTRAR EL OBJETO DE ARTE LAMADO NINO DE LA PASION	3 seg	8 seg	7 seg
	MOSTRAR EL OBJETO DE ARTE DENOMINADO INMACULADA CONCEPCION	1 seg	11 seg	9 seg
Buscar Obras u Objetos de Arte por un siglo específico	LISTAR LAS OBRASDE ARTE DEL SIGLO DIECIOCHO	4 seg	10 seg	10 seg
	MOSTRAR TODOS LOS OBJETOS DE ARTE DEL SIGLO DIECIOCHO	3 seg	10 seg	10 seg

Buscar Obras u Objetos de Arte por una técnica específica	LISTAR LAS OBRAS DE ARTE QUE ESTÁN REALIZADAS CON LA TECNICA OLEO SOBRE LIENZO	3 seg	17 seg	13 seg
	MOSTRAR LOS OBJETOS DE ARTE QUE ESTÁN ELABORADOS CON LA TECNICA TALLADO SOBRE MADERA	3 seg	18 seg	13 seg
Buscar Obras de Arte por un autor específico	LISTAR LAS OBRAS DE ARTE DEL AUTOR JOSE CORTES	3 seg	14 seg	11 seg
	MOSTAR LAS OBRAS DE ARTE DEL PINTOR PIARRETTA Y PITTERI	4 seg	11 seg	9 seg
Buscar Objetos de Arte por un autor específico	LISTAR LOS OBJETOS DE ARTE DEL ARTISTA BERNARDO DE LEGARDA	3 seg	14 seg	10 seg
	MOSTRAR LOS OBJETOS DE ARTE DEL AUTOR TALLADOR ANONIMO	3 seg	14 seg	11 seg

Tabla 41. Medición de tiempos de latencia en los tres dispositivos diferentes

De los anteriores resultados se deduce que la capacidad de procesamiento y memoria interna del dispositivo cliente sobre el cual se ejecuta la aplicación, influye en la interacción con el usuario, demostrando que a mejores especificaciones del teléfono, el tiempo de latencia es menor y por ende se produce una mejora en la experiencia de usuario. Debido a que el reconocedor se encuentra embebido en el dispositivo, la carga de las librerías y el proceso del reconocimiento, requiere mayores recursos y por ende causan degradación en teléfonos de gama menor.

4.5.1.3 Evaluación de la correspondencia comando de voz – respuesta

Para evaluar la correspondencia entre lo que el usuario consulta y la respuesta que trae consigo el servidor, se tomó como base, los resultados obtenidos tras realizar manualmente la consulta SQWRL correspondiente a la consulta de voz en el Editor Protégé 3.4.7, para compararlos con la lista obtenida como respuesta a la consulta realizada en el dispositivo.

Se realizaron una serie de pruebas consistentes en la elección de una de las posibles frases a reconocer para un mismo tipo de consulta de voz de usuario, y se comparan las gráficas obtenidas en el Editor Protégé 3.4.7 con la interfaz que muestra la lista de resultados en el dispositivo.

En la siguiente tabla se muestran las consultas SWQRL que se deben realizar a la ontología:

	Consulta
Q1	Seleccionar el Nombre, Periodo, Técnica Artística, y Autor de todas(os) Obras/Objetos de Arte.
Q2	Seleccionar el Nombre, Periodo, Técnica Artística, y Autor de la

	Obra/Objeto de Arte con un nombre específico.
Q3	Seleccionar el Nombre, Periodo, Técnica Artística, y Autor de las Obras/Objetos de Arte pertenecientes a un determinado siglo.
Q4	Seleccionar el Nombre, Periodo, Técnica Artística, y Autor de las Obras/Objetos de Arte que estén realizadas con una determinada técnica artística.
Q5	Seleccionar el Nombre, Periodo, Técnica Artística, y Autor de las Obras/Objetos de Arte pertenecientes a un determinado autor.

Tabla 42. Conjunto de consultas realizadas a la ontología.

1. Tipo de Consulta de voz: Consultar todas las Obras u Objetos de Arte

   Frase a reconocer: LISTAR TODOS LOS OBJETOS DE ARTE

   Consulta a la ontología correspondiente: Q1: Seleccionar el Nombre, Periodo, Técnica Artística, y Autor de todas(os) Obras/Objetos de Arte.

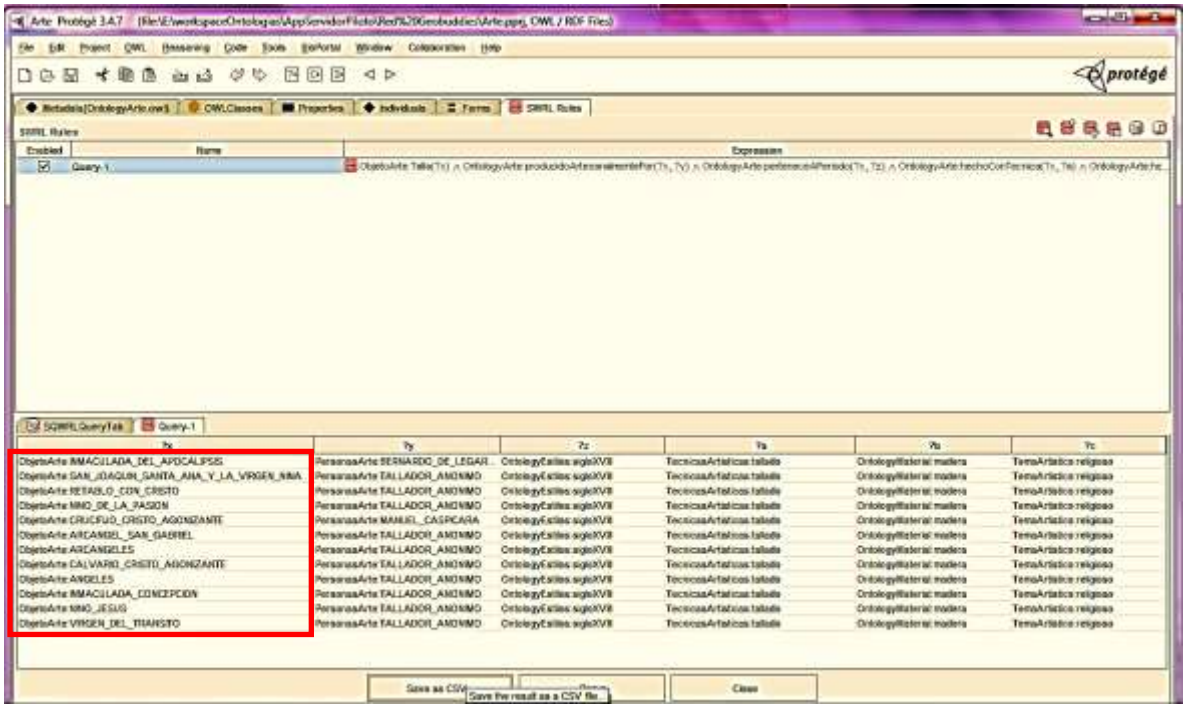

Figura 17. Resultados obtenidos de la consulta Q1 SQWRL en Protégé 3.4.7

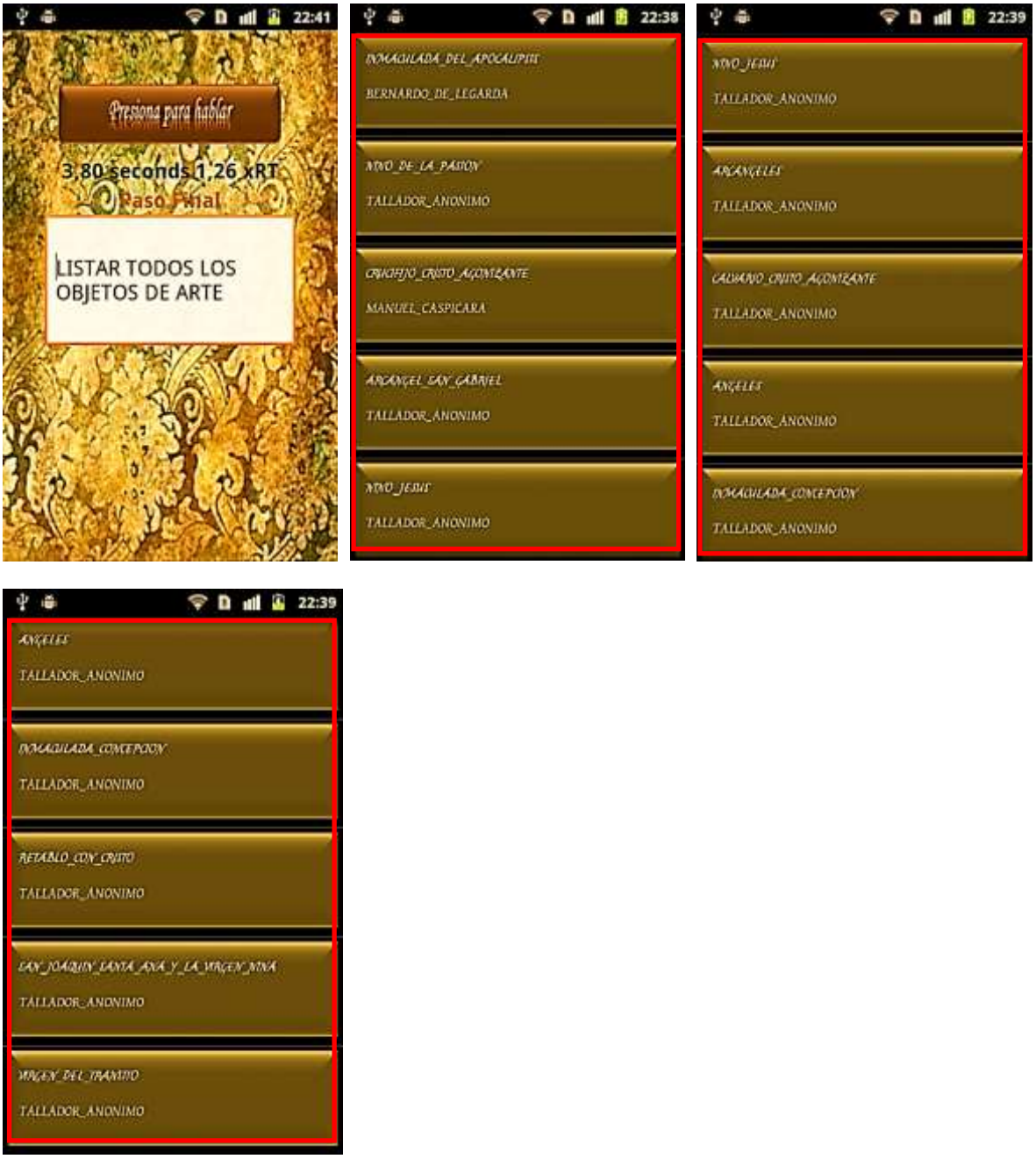

Figura 18. Lista de Resultados obtenidos de la consulta de voz No. 1 en el dispositivo

La interfaz del dispositivo (Figura 18) muestra que la totalidad de los resultados obtenidos como respuesta a la consulta por voz “Consultar todos los objetos de arte”, son los mismos que se obienen tras realizar manualmente la consulta SQWRL Q1: Seleccionar el Nombre, Periodo, Técnica Artística, y Autor de todos Objetos de Arte en el Editor Protégé 3.4.7 (Figura 17), que son en total 12.

2. Tipo de Consulta de voz: Consultar las Obras de Arte especificando el nombre
   Frase a reconocer: MOSTRAR LA OBRA DE ARTE CON NOMBRE ADORACION CINCO RAZAS

Consulta a la ontología correspondiente: Q2: Seleccionar el Nombre, Periodo, Técnica Artística, y Autor de la Obra/Objeto de Arte con un nombre específico.

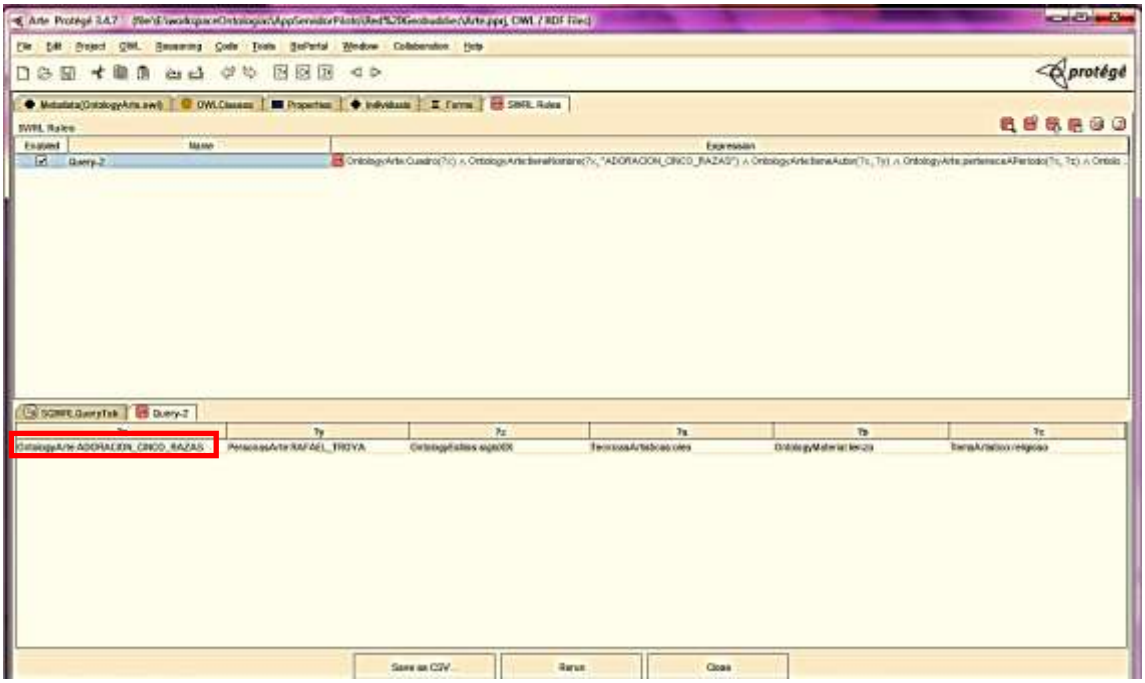

Figura 19. Resultados obtenidos de la consulta Q2 SQWRL en Protégé 3.4.7

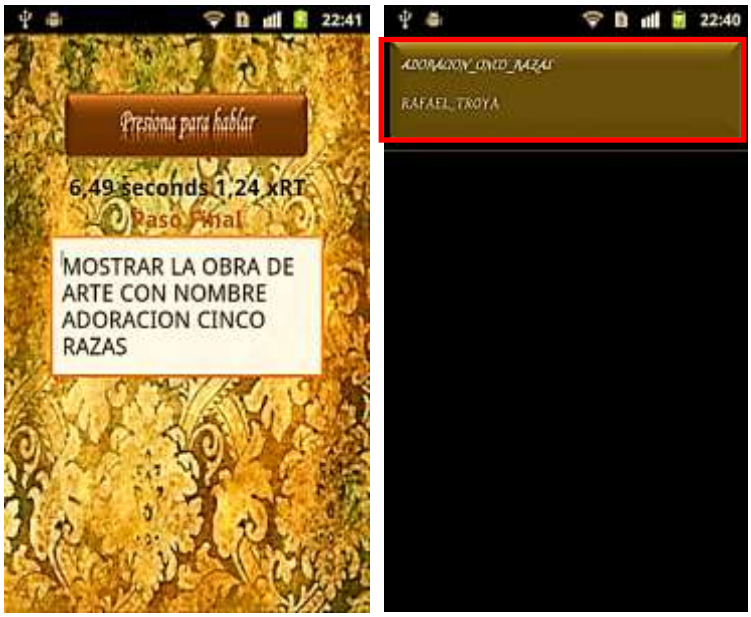

Figura 20. Lista de Resultados obtenidos de la consulta de voz No. 2 en el dispositivo

Al realizar manualmente la consulta SWQRL Q2: Seleccionar el Nombre, Periodo, Técnica Artística, y Autor de la Obra de Arte con un nombre específico en Protégé 3.4.7, en este caso “ADORACION CINCO RAZAS” muestra un único resultado como se observa en la Figura 19, el cual corresponde al

mismo que se despliega en la interfaz del dispositivo cuando se consulta por voz “Consultar la Obra de arte con nombre Adoracion Cinco Razas” mostrada en la Figura 20.

3. Tipo de Consulta de voz: Buscar Obras u Objetos de Arte por un siglo específico
   Frase a reconocer: LISTAR LAS OBRAS DE ARTE DEL SIGLO DIECINUEVE
   Consulta a la ontología correspondiente: Q3: Seleccionar el Nombre, Periodo, Técnica Artística, y Autor de las Obras/Objetos de Arte pertenecientes a un determinado siglo.

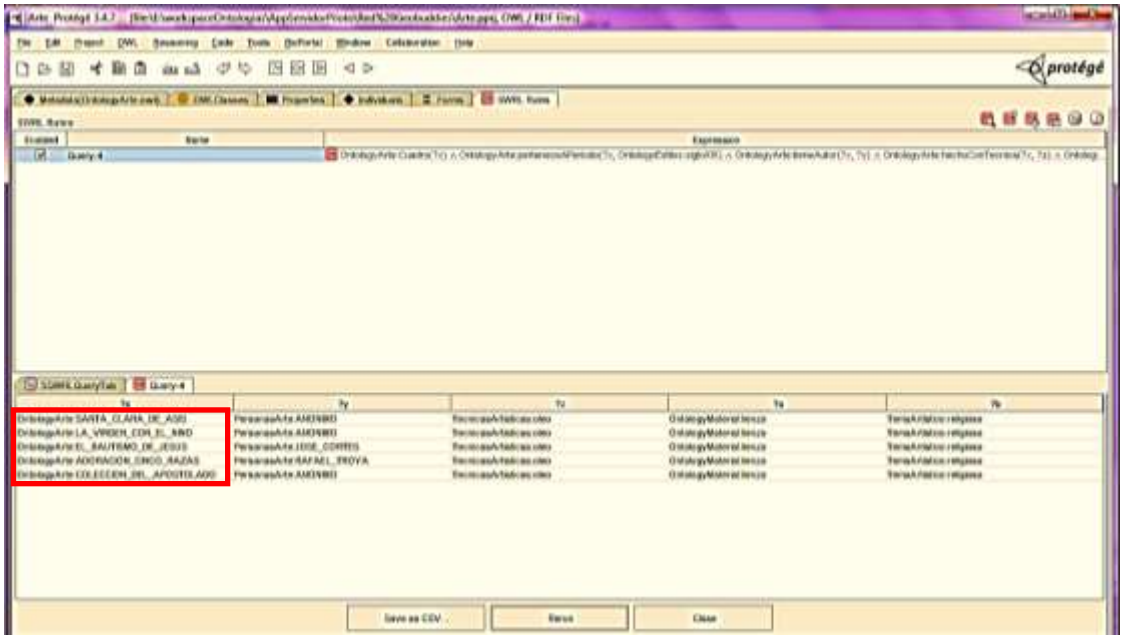

Figura 21. Resultados obtenidos de la consulta Q3 SQWRL en Protégé 3.4.7

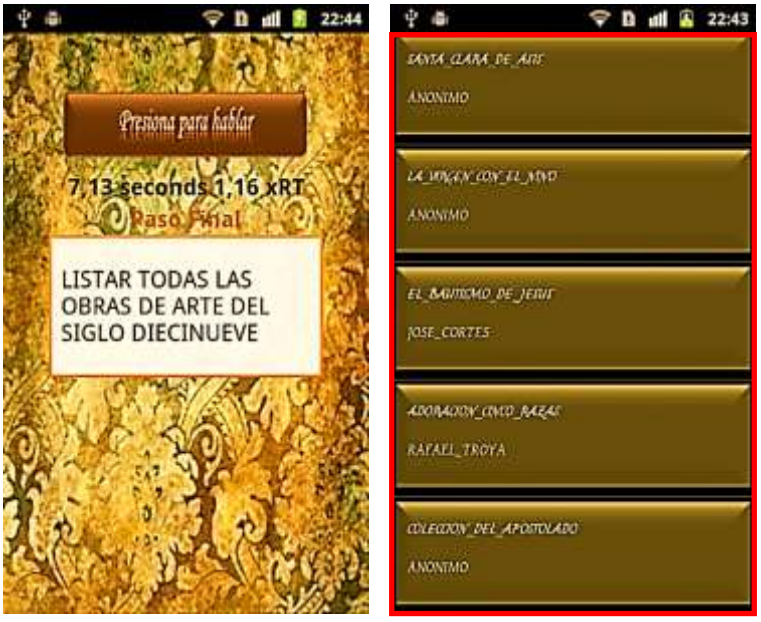

Figura 22. Lista de Resultados obtenidos de la consulta de voz No. 3 en el dispositivo

Al realizar manualmente la consulta SWQRL: Q3: Seleccionar el Nombre, Periodo, Técnica Artística, y Autor de las Obras/Objetos de Arte pertenecientes a un determinado siglo, én este caso el siglo Diecinueve en Protégé 3.4.7, muestra un total de 5 resultados, los mismos que se despliegan en la interfaz del dispositivo cuando se pronuncia la frase "Listar todas las obras de arte del siglo diecinueve" como se observa en la Figura 22.

4. Tipo de Consulta de voz: Buscar Obras u Objetos de Arte por una técnica específica
   Frase a reconocer: LISTAR LAS OBRAS DE ARTE QUE ESTÁN REALIZADAS CON LA TECNICA GRABADO SOBRE PAPEL
   Consulta a la ontología correspondiente: Q4: Seleccionar el Nombre, Periodo, Técnica Artística, y Autor de las Obras/Objetos de Arte que estén realizadas con una determinada técnica artística.

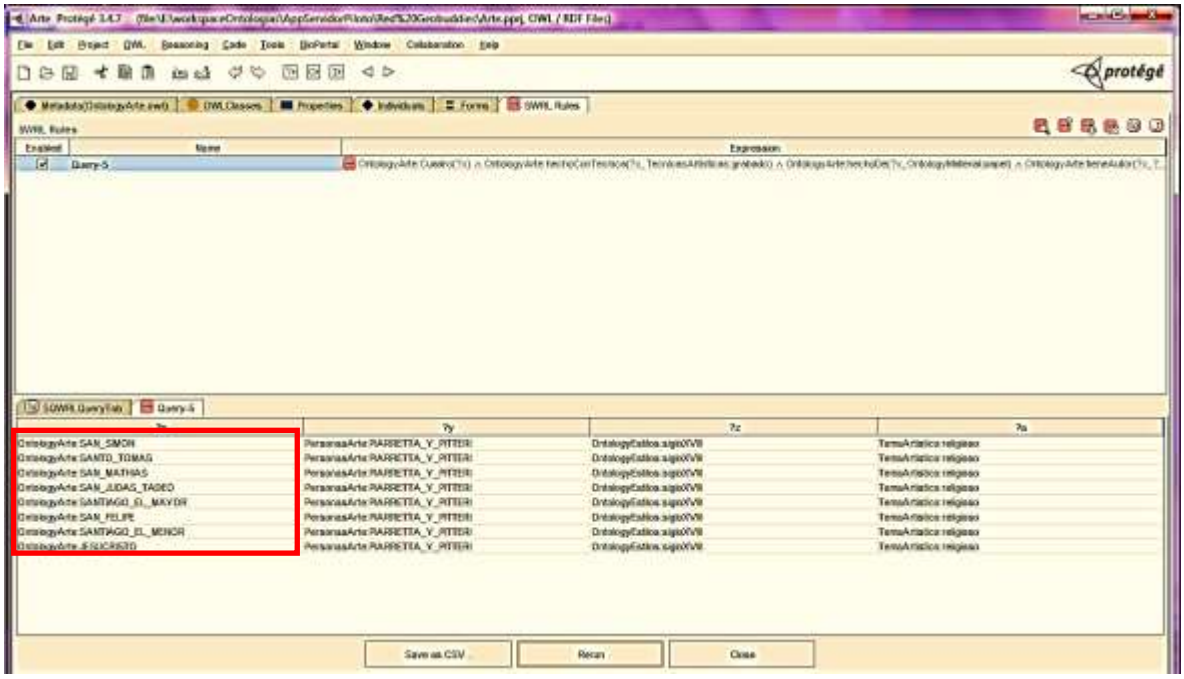

Figura 23. Resultados obtenidos de la consulta Q4 SQWRL en Protégé 3.4.7

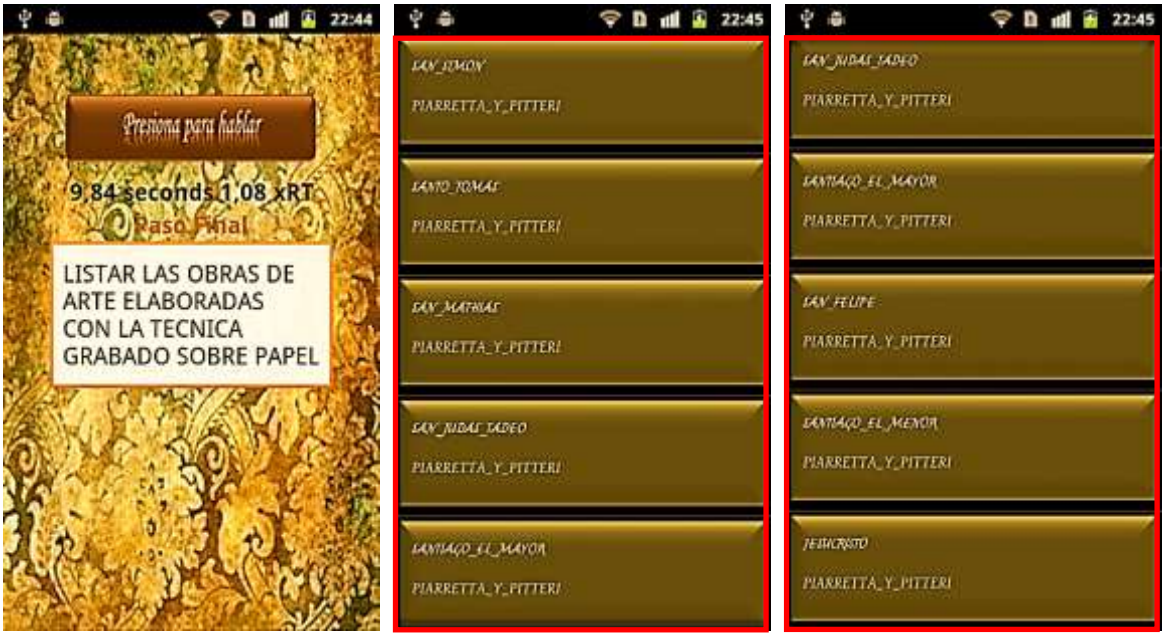

Figura 24. Lista de Resultados obtenidos de la consulta de voz No. 4 en el dispositivo

Como se pueden observar en la Figura 24, la interfaz del dispositivo muestra que la totalidad de los resultados obtenidos como respuesta a la consulta por voz "Listar las obras de arte elaboradas con la técnica grabado sobre papel" son 8, los mismos que se obtienen al realizar manualmente la consulta Q4: Seleccionar el Nombre, Periodo, Técnica Artística, y Autor de las Obras de Arte que estén realizadas con una determinada técnica artística, específicamente "Grabado sobre papel" como se observa en la Figura 23.

5. Tipo de Consulta de voz: Buscar Obras de Arte por un autor específico
   Frase a reconocer: LISTAR LAS OBRAS DE ARTE DEL AUTOR JOSE CORTES
   Consulta a la ontología correspondiente: Q5: Seleccionar el Nombre, Periodo, Técnica Artística, y Autor de las Obras/Objetos de Arte pertenecientes a un determinado autor.

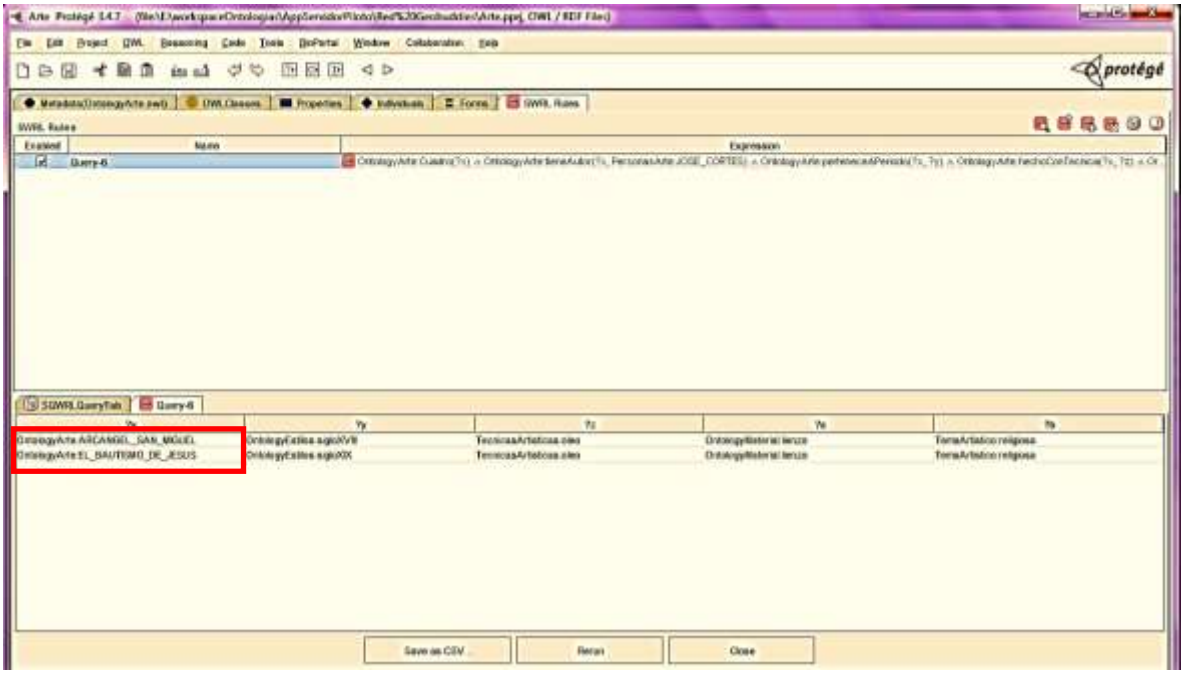

Figura 25. Resultados obtenidos de la consulta Q5 SQWRL en Protégé 3.4.7

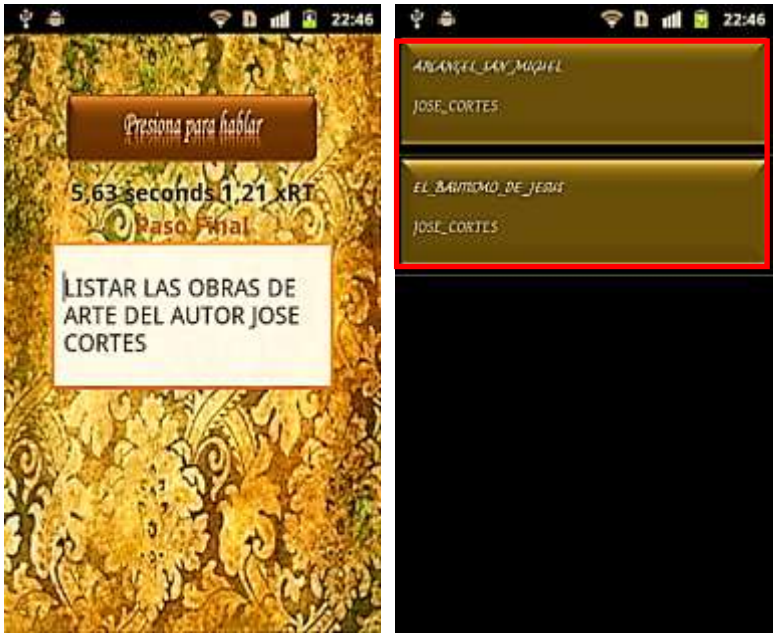

Figura 26. Lista de Resultados obtenidos de la consulta de voz No. 5 en el dispositivo

La interfaz del dispositivo (Figura 26) muestra que la totalidad de los resultados obtenidos como respuesta a la consulta por voz “Listar las obras de arte del autor Jose Cortes”, son los mismos que se obienen tras realizar manualmente la consulta SQWRL Q5: Seleccionar el Nombre, Periodo, Técnica Artística, y Autor de las Obras/Objetos de Arte pertenecientes a un determinado autor, en este caso “Jose Cortes”, en el Editor Protégé 3.4.7 (Figura 25), es decir, en total 2.

La anterior evaluación demuestra que existe una correcta correspondencia entre lo que pregunta el usuario y la consulta SQWRL que se debe hacer a la ontología, es decir las instancias obtenidas por las consultas, en su totalidad son correctas y correspondientes, como se observa en las anteriores figuras.

4.5.1.4 Evaluación de la Inferencia de nuevo conocimiento por parte de las reglas

Para evaluar la correcta inferencia de nuevo conocimiento en las instancias creadas para las Obras u Objetos de arte, es decir el Estilo de cada una de ellas, el cual no ha sido especificado explícitamente en la ontología, se incorporan de 4 reglas que indican las siguientes relaciones:

A. Todas las obras/objetos de arte que pertenecen al siglo XVI son del Estilo "arteDelRenacimiento".
B. Todas las obras/objetos de arte que pertenecen al siglo XVII son del Estilo "arteBarroco".
C. Todas las obras/objetos de arte que pertenecen al siglo XVIII son del Estilo "arteBarroco".
D. Todas las obras/objetos de arte que pertenecen al siglo XIX son del Estilo "romanticismo".

Se comparó una instancia perteneciente a cada siglo, observando en el Editor Protégé 3.4.7 la propiedad "Periodo" de cada una, verificando que no tenía ningún estilo asignado y teniendo en cuenta el "Estilo" al que debería pertenecer según las reglas establecidas, para compararlo con los detalles del mismo (obra/objeto de arte) consultado por voz en el dispositivo.

Se realizaron una serie de pruebas que consistieron en elegir una de las Obras u Objetos de arte pertenecientes a un Siglo diferente para comparar las gráficas obtenidas en el Editor Protégé 3.4.7 y la interfaz que muestra los detalles de la Obra u Objeto de arte consultada por el "Nombre" en el dispositivo.

- Obra de Arte: EL_MARTIRIO_DE_SANTA_BARBARA
  Periodo: Siglo XVII
  Estilo al que debe pertenecer: arteBarroco

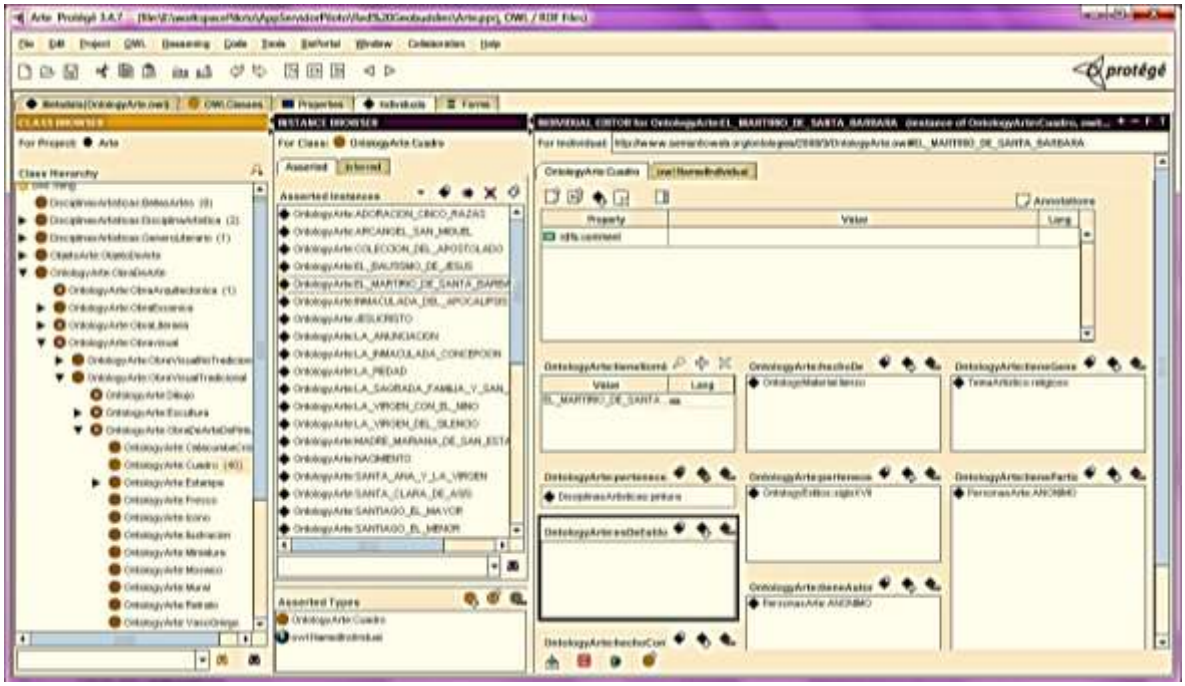

Figura 27. Propiedades de la instancia "EL_MARTIRIO_DE_SANTA_BARBARA" en Protégé 3.4.7

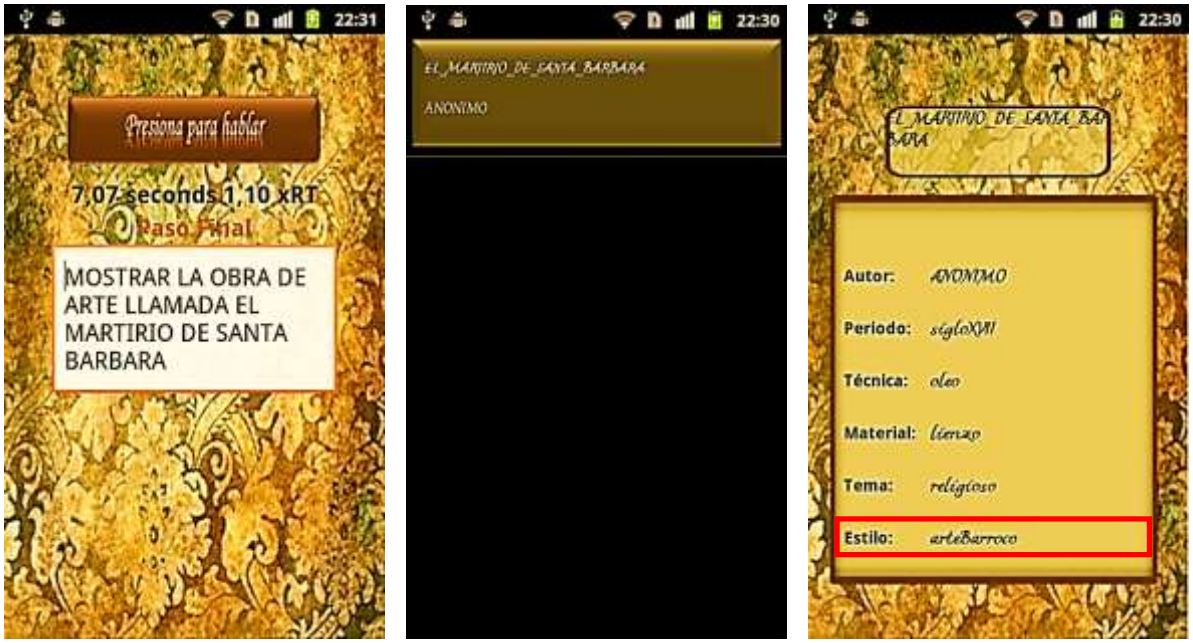

Figura 28. Detalles de la obra "EL_MARTIRIO_DE_SANTA_BARBARA" en el dispositivo

La Obra de Arte: EL_MARTIRIO_DE_SANTA_BARBARA indicada en la Figura 27, pertenece al siglo XVII y según la regla B, el estilo al que debe pertenecer es "arteBarroco", el cual no se encuentra especificado como propiedad, mientras que la Figura 28, muestra la interfaz de detalle de la obra consultada "EL MARTIRIO DE SANTA BARBARA", y en ella se encuentra el Estilo inferido "arteBarroco".

- Obra de Arte: INMACULADA_DEL_APOCALIPSIS
  Periodo: Siglo XVIII
  Estilo al que debe pertenecer: arteBarroco

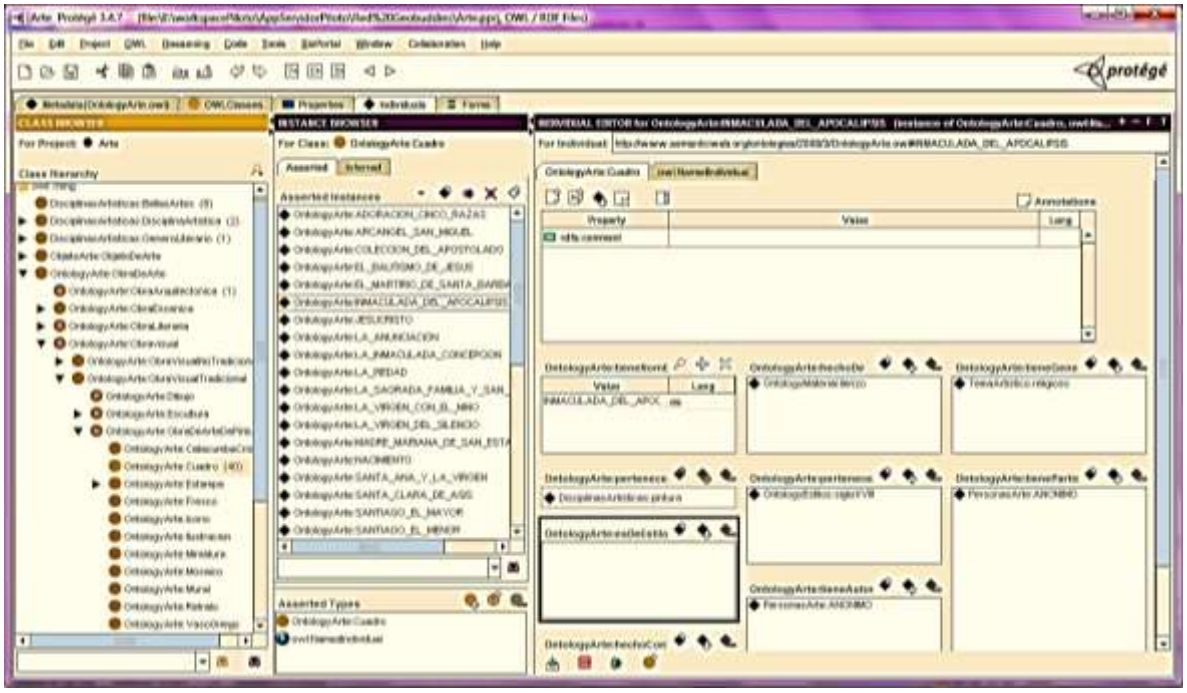

Figura 29. Propiedades de la instancia "INMACULADA_DEL_APOCALIPSIS" en Protégé 3.4.7

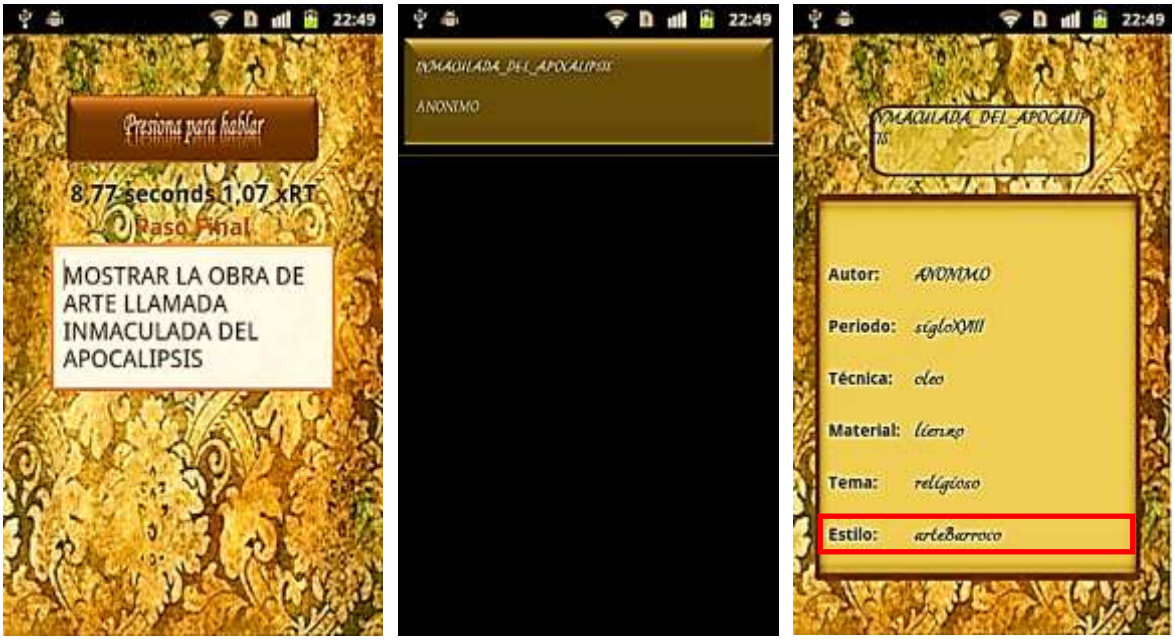

Figura 30. Detalles de la obra "INMACULADA_DEL_APOCALIPSIS" en el dispositivo

La Figura 30, muestra la interfaz de detalle de la obra consultada "INMACULADA_DEL_APOCALIPSIS", y el Estilo al que pertenece "arteBarroco", el cual no se encuentra especificado como propiedad en la Obra de Arte: INMACULADA_DEL_APOCALIPSIS,

pero pertenece al siglo XVIII y según la regla C, el estilo al que debe pertenecer es “arteBarroco”, como se observa en la Figura 29.

- Obra de Arte: COLECCIÓN_DEL_APOSTOLADO
  Periodo: Siglo XIX
  Estilo al que debe pertenecer: romanticismo

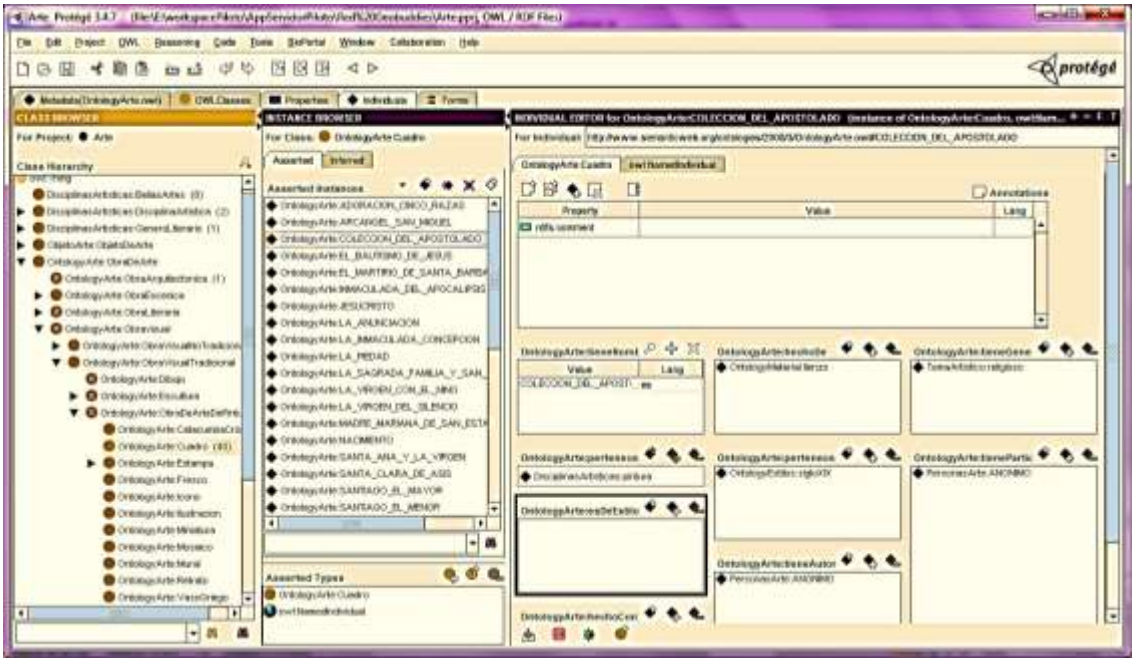

Figura 31. Propiedades de la instancia “COLECCIÓN_DEL_APOSTOLADO” en Protégé 3.4.7

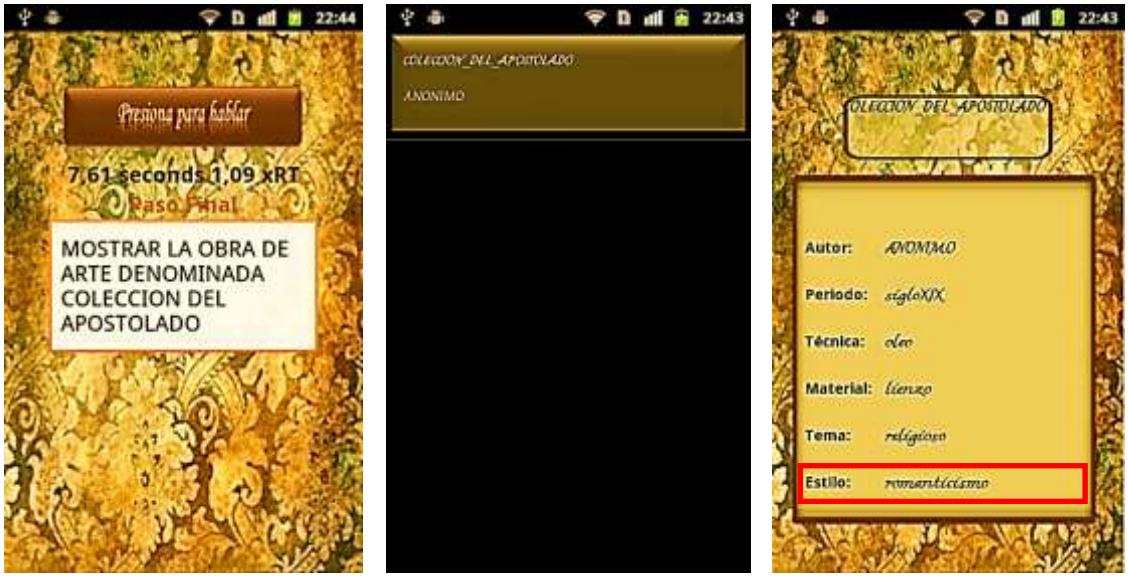

Figura 32. Detalles de la obra “COLECCIÓN_DEL_APOSTOLADO” en el dispositivo

La Figura 31, indica que la Obra de Arte: EL_ COLECCIÓN_DEL_APOSTOLADO, pertenece al siglo XIX y según la regla D, el estilo al que debe pertenecer es “romanticismo”, el cual no se

encuentra especificado como propiedad, mientras que la Figura 32, muestra la interfaz de detalle de la obra consultada “EL_ COLECCIÓN_DEL_APOSTOLADO”, si se encuentra el Estilo inferido “romanticismo”.

Finalmente, gracias a la última evaluación se comprobó que gracias a la inclusión de las reglas se puede obtener nuevo conocimiento, es decir inferir información que no se encuentra especificada explícitamente en la ontología, en este caso específico, se infiere el Estilo al que pertenece cada Obra u Objeto de arte a partir de las reglas establecidas.

CAPÍTULO 5

APORTES, CONCLUSIONES Y TRABAJOS FUTUROS

5.1 APORTES DEL TRABAJO DE GRADO

Se generó un mecanismo de procesamiento de la gramática de contexto para los desarrolladores de aplicaciones móviles basadas en voz sobre la plataforma Android, que deseen utilizar ontologías de dominio para representar el vocabulario o gramática específica de los ASR a utilizar para el reconocimiento de voz. Específicamente, se presentó una evaluación de los módulos requeridos para dicho procesamiento, para determinar cuáles fueron los motores ASR Open Source que presentaron mejores capacidades de configuración sobre la plataforma Android, y qué mecanismo de razonamiento ontológico ofreció un mejor procesamiento de dicha gramática, por medio de la definición de algunos criterios de evaluación comparados con el desempeño de las herramientas software que implementan tales módulos.

Se indicaron las capacidades de configuración y reconocimiento de voz de los motores ASR Open Source evaluados y se seleccionó aquel que presentó las mejores capacidades sobre la plataforma Android. Se adaptó un mecanismo de razonamiento ontológico a través de la exploración de métodos de razonamiento de ontologías existentes, combinando diversos componentes predominantes: un razonador, un motor de reglas, un lenguaje de consulta y una API de programación para ontologías.

En este sentido, los anteriores módulos conforman la arquitectura base definida, que se encarga de realizar el procesamiento de las frases pronunciadas por el usuario a la interfaz de consulta de voz de la aplicación móvil, permitiendo a los desarrolladores basarse en este mecanismo de razonamiento de la información asociada a un dominio específico.

Se creó entonces, un piloto con una aplicación móvil basada en consultas por voz para un contexto específico, que permitiera realizar la evaluación de diferentes métricas de reconocimiento de voz y procesamiento del mecanismo creado en un entorno real, por medio de diferentes consultas por voz. Se realizó a su vez, el análisis de los resultados obtenidos, los cuales fueron documentados y tras establecer las conclusiones necesarias se dió respuesta a la pregunta de investigación: ¿Cómo procesar la gramática de contexto de una aplicación móvil basada en voz para clientes Android usando ontologías de dominio?

Finalmente, algunos trabajos relacionados sobre sistemas operativos como iPhone, Symbian, Blackberry o Windows Phone, realizan aportes interesantes al tema del tratamiento de aplicaciones móviles basadas en voz con especificación de gramáticas por medio de ontologías, pero al no existir una previa exploración de dicha temática en una de las plataformas más difundidas actualmente para los teléfonos inteligentes, como lo es Android, este trabajo de grado, realiza su principal aporte investigativo al respecto, estableciendo conclusiones significativas sobre el uso de ontologías de dominio aplicadas al reconocimiento de voz sobre dicha plataforma, que permitan plantear un camino alternativo en la evolución de estos sistemas.

Adicionalmente se generó un Artículo denominado “Tools Evaluation for Domain Ontologies Based Speech Recognition over Android Platform”, el cual se encuentra sometido a evaluación para el evento 2012 IEEE COLOMBIAN COMMUNICATIONS CONFERENCE (COLCOM 2012).

5.2 CONCLUSIONES

En el marco de las aplicaciones móviles basadas en voz utilizando ontologías de domino como gramática de contexto, se debe tener en cuenta que para definir y configurar una gramática propia es de vital importancia emplear un motor de reconocimiento que permita realizar tal configuración y emplear un mecanismo que permita razonar la ontología de dominio particular elegida. Al describir la gramática de contexto para una determinada aplicación móvil basada en voz por medio de ontologías de dominio, no sólo se restringe el lenguaje de entrada al reconocedor de voz, sino también, se ofrece una mayor precisión en el proceso de reconocimiento y la obtención de información consistente.

A través del estudio de los motores ASR de libre distribución, junto con sus capacidades de reconocimiento de voz y configuración; y del estudio de los mecanismos de razonamiento ontológicos que se pueden aplicar, se definió un mecanismo que permita el procesamiento de la gramática de contexto de una aplicación móvil Android basada en voz, el cual consta de la definición e integración de los módulos involucrados: los motores ASR de libre distribución y los componentes de razonamiento ontológico.

Para la determinación de los componentes de razonamiento ontológico se realizó una adaptación a partir de los existentes, combinando algunos módulos de uso frecuente en los trabajos relacionados con el razonamiento [113 , 116, 134-136]. Epecíficamente, fueron definidos los siguientes componentes: un razonador, un lenguaje de reglas, un lenguaje de consulta (y sus motores de ejecución respectivos) y una API de programación de ontologías.

Las pruebas realizadas a los anteriores módulos permitieron concluir que en el marco de los motores de reconocimiento de voz de libre distribución y para un contexto en el idioma español, es deseable trabajar con PocketSphinx, ya que a pesar de que Julius ofrece la capacidad para configurar la gramática de contexto, sólo lo permite para el idioma inglés. Por otro lado el API de reconocimiento de Android a pesar del amplio soporte de idiomas y la cantidad de palabras que reconoce, no permite la configuración de una gramática específica, lo cual se convierte en un gran limitante para propósitos de extensión y personalización de los motores ASR; en contraste, PocketSphinx no sólo permite la configuración de la gramática de contexto sino que demuestra un rendimiento superior con respecto a los otros motores sometidos a evaluación.

En el marco de los componentes de razonamiento ontológico, como razonador, Pellet demostró ser el más eficaz y rápido por la precisión en sus resultados, como lenguaje de reglas, SWRL posee la ventaja de permitir la adición de reglas en código y su debida ejecución por medio del sistema experto de reglas Jess, como lenguaje de consultas, SQWRL opera muy bien en conjunto con las reglas SWRL, y es especialista en consultas a ontologías OWL gracias a su enfoque semántico obteniendo todos los resultados esperados, y finalmente como API de Programación, Protégé OWL API soporta los componentes antes mencionados, además es un API de alto nivel y utiliza e incorpora varias funcionalidades de Jena por debajo, como por ejemplo cargar modelos en memoria.

En este sentido, la arquitectura que se conformó fue tipo cliente-servidor, ya que los cálculos de razonamiento complejo, en los que se derivan nuevo conocimiento a partir de una ontología existente, sólo se pueden realizar en el lado del servidor, ya que se requiere una alta capacidad de procesamiento; por esta razón, las tareas de reconocimiento de voz se dejaron en el móvil.

Por otro lado, en la utilización de ontologías creadas por terceros es importante referirse a la documentación para conocer la metodología con la cual la ontología fue creada, para establecer así, nuevos criterios que permitan incrementar la confiabilidad en los resultados provenientes de la evaluación que se realice. Así por ejemplo, para el caso de la ontología "Arte.owl", conocer cuál fue la metodología sobre la cual se construyó, es decir *NeOn*, de alguna manera permitió garantizar que el desarrollo se había realizado adecuadamente.

La implementación y pruebas del piloto construido permitieron observar que la tasa correcta de reconocimiento de voz de las frases programadas en la aplicación es alta, lo que demuestra un buen desempeño para el reconocedor elegido, y que existe una correcta correspondencia entre lo que pregunta el usuario y la consulta SQWRL que se debe hacer a la ontología. Por otro lado, la capacidad de procesamiento y memoria interna del dispositivo cliente sobre el cual se ejecuta la aplicación, influye en la experiencia de usuario, ya que a mejores capacidades del teléfono, la latencia es menor y mejora la percepción por parte del usuario final. Finalmente, la inclusión de reglas a la aplicación demostró su eficacia en la inferencia de conocimiento que no se encontraba especificado explícitamente en la ontología de dominio elegida.

No obstante, no todas las frases se reconocieron correctamente, lo cual condujo a la repetición de las consultas para su correcto reconocimiento. Esto se debe a que se genera ambigüedad en el reconocimiento cuando la extensión de la gramática es grande, sobre todo, en las ultimas consultas que en que se encuentren programadas.

5.3 TRABAJOS FUTUROS

Como trabajo futuro, en el contexto de la representación de la gramática de un reconocedor de voz, se propone definir un "formato genérico" de gramática que utilice ontologías de dominio, que permita representar el vocabulario o gramática específica del contexto, en el cual se enmarca una aplicación móvil basada en voz y que pueda ser fácilmente adaptado por terceros a otros contextos.

Por otro lado, en el marco de la interacción con el usuario y las posibilidades de sofisticación de la aplicación móvil basada en voz, se propone crear una ontología de dominio propia en el idioma español y no emplear una ya existente, la cual cuente con toda la información necesaria a utilizar, y a partir de la misma, especificar múltiples gramáticas de contexto del reconocedor de voz que permitan de igual manera, diversas opciones de interactividad en la aplicación y de su extensión dependerá la cantidad de obtención de información consistente.

REFERENCIAS

[1] A. Toninelli, *et al.*, "Middleware support for mobile social ecosystems," in *IEEE 34th Computer Software and Applications Conference Workshops*, 2010, pp. 293-298.

[2] K. Yee, et al., "OntoMobiLe: a generic ontology-centric service-oriented architecture for mobile learning," in *Tenth International Conference on Mobile Data Management: Systems, Services and Middleware*, 2009.

[3] D. Sonntag, "Context-Sensitive Multimodal Mobile Interfaces," in *9th International Conference on Human Computer Interaction with Mobile Devices and Services*, 2007.

[4] K. Lee and R. Grice, "The Design and Development of User Interfaces for Voice Application in Mobile Devices," in *Proceedings of the IEEE International Professional Communication Conference*, 2007, pp. 308-320.

[5] M. Avvenuti and A. Vecchio, "Mobile visual access to legacy voice-based applications," in *Proceedings of the 6th International Conference on Mobile Technology, Applications and Systems (ACM Mobility)*, 2009, pp. 1-6.

[6] W. Mueller, *et al.*, "Interactive multimodal user interfaces for mobile devices," in *Proceedings of the 37th Annual Hawaii International Conference on System Sciences*, 2004, p. 10.

[7] J. Louw, "Speect: a multilingual text-to-speech system," in *Proceedings of the 19th Annual Symposium of the Pattern Recognition Association of South Africa (PRASA)*,, 2008, pp. 165–168.

[8] K. Wongpatikaseree, *et al.*, "A hybrid diphone speech unit and a speech corpus construction technique for a Thai text-to-speech system on mobile devices," in *International Conference of Electrical Engineering/Electronics Computer Telecommunications and Information Technology (ECTI-CON)*, 2010, pp. 1089-1093.

[9] S. Thomas, *et al.*, "Distributed Text to Speech Synthesis for Embedded Systems–An analysis," in *The National Conference on Communications NCC2005*, 2004, pp. 273-276.

[10] W. Kurschl, *et al.*, "Development Issues for Speech-Enabled Mobile Applications," in *Proceedings of Software Engineering, Fachtagung des GI-Fachbereichs Softwaretechnik*, 2007.

[11] S. A. Ramakrishnan IV., Yang G., "Hearsay: enabling audio browsing on hypertext content," in *Proceedings of the 13th international conference on World Wide Web*, 2004, p. 89.

[12] V. Zue, *et al.*, "JUPITER: A telephone-based conversational interface for weather information," in *IEEE Transactions on Speech and Audio Processing*, 2002, pp. 85-96.

[13] K. S. Leong L. H., Koshizuka N., Sakamura K., "CASIS: A context-aware speech interface system," in *Proceedings of the 10th international conference on Intelligent user interfaces*, 2005, pp. 231-238.

[14] S. Oviatt, "Advances in Robust Multimodal Interface Design," in *IEEE Computer Graphics and Applications*, 2003, pp. 62-68.

[15] N. Pfleger, "Context based multimodal fusion," in *Proceedings of the 6th international conference on Multimodal interfaces*, 2004, pp. 265-272.

[16] M. Z. Kopsa J., Slavik P., "Ontology Driven Voice-based Interaction in Mobile Environment," in *Proceedings of Mobile Computing and Ambient Intelligence: The Challenge of Multimedia*, 2005.

[17] W. D. Massie Thomas, Leo Obrst, "TVIS: Tactical Voice Interaction Services for Dismounted Urban Operations," in *Military Communications Conference MILCOM, IEEE*, George Mason University, 2009, pp. 1-7.

[18] Android Developers, "Speech Input API for Android," *disponible en: http://developer.android.com/resources/articles/speech-input.html. Consultado el 20 de Mayo de 2011.*

[19] J. Ryan, "The market for speech applications in mobile computing expected to triple by 2014," in *Published by Datamonitor. Disponible en: http://about.datamonitor.com/media/archives/2649*, ed. Consultado el 6 de Abril de 2011. , 20 May 2009.
[20] A. Neustein, *Advances in Speech Recognition: Mobile Environments, Call Centers and Clinics*. New York: Springer, 2010.
[21] H. M. Kosinowski, "Modular Grammars for Speech Recognition in Ontology-Based Dialogue Systems," Faculty of Computational Linguistics, Saarland, 2010.
[22] L. R. Rabiner and B. Juang, "Statistical methods for the recognition and understanding of speech," in *Encyclopedia of language and linguistics*, ed, 2004.
[23] L. R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," in *Proceedings of the IEEE*, 1989, pp. 257–286.
[24] L. R. Rabiner, & Juang, Biing-Hwang, *Fundamentals of Speech Recognition*: NJ: Prentice Hall, 1993.
[25] X. Huang, Acero, Alex, & Hon, Hsiao-Wuen, *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. Upper Saddle River, NJ: USA: Prentice Hall PTR, 2001.
[26] D. Jurafsky, & Martin, James H., *Speech and Language Processing*: Prentice Hall, 2008.
[27] S. E. Levinson, Rabiner L.R., and Sondhi, M.M., *An Introduction to the Application of the Theory of Probabilistic Functions of a Markov Process to Automatic Speech Recognition* vol. 62, 1983.
[28] J. D. Ferguson, *Hidden Markov Analysis: An Introduction, in Hidden Markov Models for Speech*. Princeton, NJ: Institute for Defense Analyses, 1980.
[29] L. R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," in *Proceedings of the IEEE*, 1989.
[30] L. R. Rabiner, and Juang, B.H., "An Introduction to Hidden Markov Models," in *IEEE Signal Processing Magazine*, 1985.
[31] B. H. Juang, Levinson, S.E., and Sondhi, M.M, "Maximum Likelihood Estimation for Multivariate Mixture Observations of Markov Chains," in *IEEE Trans. Information Theory*, 1986, pp. 307-309.
[32] B. H. Juang, "Maximum Likelihood Estimation for Mixture Multivariate Stochastic Observations of Markov Chains," in *AT&T Tech. J*, 1985, pp. 1235-1249.
[33] Microsoft Research, "Acoustic Modeling," *Disponible en: http://research.microsoft.com/en-us/projects/acoustic-modeling/. Consultado el 21 de enero de 2012.*
[34] M. D. Riley, et al., "Stochastic Pronunciation Modeling from Hnad-Labelled Phonetic Corpora," *Speech Communication*, 1999.
[35] R. Rosenfeld, "Two Decades of Statistical Language Modeling: Where Do we Go From Here?," in *Proceedings of the IEEE*, Special Issue on Spoken Language Processing, 2000.
[36] F. Jelinek, Mercer, R.L., and Roukos, S., *Principles of lexical language modeling for speech recognition, in Advances in Speech Signal Processing*. New York, 1991.
[37] A. Bhushan, "Top Best Voice or speech Recognition Software," *Disponible en: http://ceoworld.biz/ceo/2010/01/25/top-best-voice-or-speech-recognition-software. Consultado el 20 de enero de 2012.*, 2010.
[38] N. H. H. Kokubo, A. Lee, T. Kawahara and K. Shikano, "Real-Time Continuous Speech Recognition System on SH-4A Microprocessor," in *Proc. International Workshop on Multimedia Signal Processing (MMSP)*, 2007, pp. 35-38.
[39] "Vocalia 2.2 Speech Recognition for the iPhone,," *Disponible en: http://www.creaceed.com/vocalia/. Consultado el 20 de enero de 2012.*
[40] T. C. e. al., "Development, Long-Term Operation and Portability of a Real-Environment Speech-oriented Guidance System," in *IEICE Trans. Information and Systems*, 2008, pp. 576–587.

[41] H. N. T. Kawahara, T. Shinozaki and S. Furui, "Benchmark Test for Speech Recognition Using the Corpus of Spontaneous Japanese.," in *Proc. ISCA & IEEE Workshop on Spontaneous Speech Processing and Recognition (SSPR)*, 2003.
[42] O. M. D. Fohr, C. Cerisara and I. Illina, "The Automatic News Transcription System: ANTS, some Real Time Experiments," in *Proc. INTERSPEECH*, 2004, pp. 377–380.
[43] K. I. a. S. F. D. Yang, "Accent Analysis for Mandarin Large Vocabulary Continuous Speech Recognition.," in *EICE Technical Report, Asian Workshop on Speech Science and Technology*, 2003, pp. 87–91.
[44] C. W. M. Jongtaveesataporn, K. Iwano and S. Furui, "Development of a Thai Broadcast News Corpus and an LVCSR System," in *ASJ Annual meeting*, 2008.
[45] T. Alumae., "Large Vocabulary Continuous Speech Recognition for Estonian using Morphemes and Classes," in *Proc. ICSLP*, 2004, pp. 389-392.
[46] M. S. M. T. Rotovnik, B. Horvat and Z. Kacic, "A Comparison of HTK, ISIP and Julius in Slovenian Large Vocabulary Continuous Speech Recognition.," in *Proc. ICSLP*, 2002, pp. 671–684.
[47] H.-Y. J. J.-G. Kim, H.-Y. Chung., "A Keyword Spotting Approach Based on Pseudo N-Gram Language Model," in *Proc. SPECOM*, 2004, pp. 256–259.
[48] A. Lee and T. Kawahara, "Recent Development of Open-Source Speech Recognition Engine Julius," in *Proceedings Asia-Pacific Signal and Information Processing Association*, Annual Summit and Conference, 2009.
[49] T. K. A.Lee, K.Takeda and K.Shikano, "A New Phonetic Tied-Mixture Model for Efficient Decoding," in *Proc. IEEE-ICASSP*, 2000, pp. 1269–1272.
[50] T. M. K. Tokuda, N. Miyazaki and T. Koba, "Hidden Markov Models Based on Multi-Space Probability Distribution for Pitch Pattern Modeling," in *Proc. IEEE-ICASSP*, 1999, pp. 229–232.
[51] "HMM-based Speech Synthesis System (HTS)," *Disponible en: http://hts.sp.nitech.ac.jp/. Consultado el 23 de enero de 2012.*, 2011.
[52] "HTK," *Disponible en: http://htk.eng.cam.ac.uk/. Consultado el 23 de enero de 2012.*
[53] "PocketSphinx 0.6 release," *Disponible el: http://cmusphinx.sourceforge.net/2010/03/pocketsphinx-0-6-release/. Consultado el 23 de enero de 2012.*
[54] "Basic concepts of speech," *Disponible en: http://cmusphinx.sourceforge.net/wiki/tutorialconcepts?s[]=acoustic&s[]=model#models. Consultado el 23 de enero de 2012.*
[55] D. Huggins-Daines, "CMU Sphinx Open Source Models," *Disponible en: http://www.speech.cs.cmu.edu/sphinx/models/. Consultado el 23 de enero de 2012.*, 2008.
[56] "CMU Sphinx FAQ," *http://www.speech.cs.cmu.edu/sphinx/doc/sphinx-FAQ.html*.
[57] C. M. University, "Building Language Model," *Disponible en: http://cmusphinx.sourceforge.net/wiki/tutoriallm?s=language&s=tool. Consultado el 27 de Octubre de 2011.*
[58] "JSpeech Grammar Format," 2000, *Disponible en: http://www.w3.org/TR/jsgf/. Consultado el 1 de noviembre de 2011.*
[59] N. Guarino, "Formal Ontology and Information Systems," in *Proceedings of Formal Ontology in Information Systems*, 1998.
[60] M. Klein, et al., *Ontologies and Schema Languages on the Web, in Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*: MIT Press, 2002.
[61] H. A. F. Fernández, "Construcción de ontologías OWL," Publicación en la Revista Vínculos Vol. 4 No. 1, Universidad Distrital Francisco José de Caldas, Facultad Tecnológica ,2009.
[62] E. R. y. H. Nuñez, "ONTOLOGÍAS: componentes, metodologías, lenguajes, herramientas y aplicaciones," Publicacion en la revista Lecturas en Ciencias de la Computación , Universidad Central de Venezuela, Facultad de Ciencias, Escuela de Computación, Caracas, 2007.

[63] A. Lozano Tello, "Métrica de idoneidad de ontologías," Tesis Doctoral, Departamento de Informática, Universidad de Extremadura, Escuela Politécnica de Cáceres, España, 2002.
[64] F.-L. M. a. Gómez-Pérez A, *Ontological Engineering*: Springer Verlag London, 2004.
[65] R. L. Roberto, "Especificación OWL de una ontología para teleeducación en WEB semántica," Tesis doctoral, Departamento de Comunicaciones, Universidad Politécnica de Valencia, 2008.
[66] J. De Bruijn, "Using Ontologies. Enabling Knowledge Sharing and Reuse on the Semantic Web," Digital Enterprise Research Institute (DERI) Technical Report DERI-2003-10-29, Austria, 2003.
[67] H. I. Fensel D, Van Harmelen F, Decker S, Erdmann M y Klein M., "OIL in a Nutshell," in *Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management*, 2000.
[68] Karp Peter, Chaudhri Vinay y Thomere Jerome, "XOL: An XML-Based Ontology Exchange Language," Versión 0.4. Technical report 559. AI Center, SRI International, 1999.
[69] S. X. e. I. Lars, "Using a Semiotic Framework for a Comparative Study of Ontology Languages and Tools," University of Science and Technology, Norway, 2005.
[70] M. D. y. V. H. Frank., "OWL Web Ontology Language Overview," *W3C Recommendation, disponible en: http://www.w3.org/TR/2004/REC-owl-features-20040210/#s1.3*, 2004.
[71] H. Knublauch, "The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications.," *Stanford University*, 2005.
[72] H. Knublauch, "Protege-OWL API Programmer's Guide," *Disponible en: http://protegewiki.stanford.edu/wiki/ProtegeOWL_API_Programmers_Guide. Consultado el 12 de noviembre de 2011.*, 2010.
[73] "THE OWL API," *Disponible en: http://owlapi.sourceforge.net/. Consultado el 23 de enero de 2012.*
[74] F. G. Sánchez, "Sistema basado en Tecnologías del Conocimiento para Entornos de Servicios Web Semánticos," Departamento de Ingeniería de la Información y las Comunicaciones, Universidad de Murcia, 2007.
[75] S. N. Russell, P., *Inteligencia Artificial. Un Enfoque Moderno.* México: Segunda edición. Ed. Pearson Prentice Hall., 2004.
[76] P. I. J. F. Vitelli, "Introducción al Razonamiento sobre Ontologías," *Publicación en Lecturas en Ciencias de la Computación, Facultad de Ciencias, Universidad Central de Venezuela*, 2011.
[77] D. C. F. Baader, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider, "The Description Logic Handbook: Theory, Implementation, Applications.," Cambridge University Press, Cambridge, UK2003.
[78] J. M. Alban Gaignard, "Survey on semantic data stores and reasoning engines," VIP Project, Milestone, 2010.
[79] P. B. Sirin E., Grau B.C., Kalyanpur A., Katz Y., "Pellet: A practical owl-dl reasoner," *Web Semantics: science, services and agents on the World Wide Web*, vol. 5, 2007, pp. 51-53.
[80] T. Briggs, "Constraint Generation and Reasoning in OWL," PhdThesis, Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County, 2008.
[81] D. B. Luis Polo, Emilio Rubiera, Sergio Fernández, "Experimento semántico para definir contextos y recursos: 1.0. ," *Proyecto MORFEO*, 2007.
[82] I. D. J. J. Carroll, C. Dollin, D. Reynolds, A. Seaborne, K. Wilkinson, "Jena: Implementing the semantic web recommendations," in *Proc. of the 13th Int. World Wide Web Conference (WWW 2004)*, 2004.
[83] P. L. S. Bechhofer, R. Volz, "Cooking the semantic web with the OWL API," in *Proc. of the 2nd Int. SemanticWeb Conf. (ISWC 2003)*, Sanibel Island, Florida, 2006.

[84] R. M. o. S. Bechhofer, P. Crowther,, "The DIG description logic interface," in *Proc. of the Int. Description Logics Workshop (DL 2003)*, 2003.
[85] K. B. S. Bechhofer. Department of ComputerScience, "FaCT++," *Disponible en: http://owl.man.ac.uk/factplusplus/. Consultado el 25 de enero de 2012.*
[86] I. d. I. d. l. U. o. I. (STI). "Racer Renamed Abox and Concept Expression Reasoner," *Disponible en: http://www.sts.tu-harburg.de/~r.f.moeller/racer/. Consultado el 25 de enero de 2012.*
[87] L. W. Huang T., Yang C., "Comparison of Ontology Reasoners: Racer, Pellet, Fact++," in *American Geophysical Union, Fall Meeting*, 2008, p. 1068.
[88] J. W. Lloyd, *Foundations of logic programming*, 2nd Edition ed.: Springer-Verlag, 1987.
[89] Claudia Milena Rodríguez A., *et al.*, "Razonadores semánticos: un estado del arte," *Publicación en la Revista de la Facultad de Ingeniería*, vol. Año 11 No. 21, 2010.
[90] Yunchuan Sun Junsheng Zhang Wei Zhao Yingjie Tian, "Managing and Refining Rule Set for SWRL," *Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference on* pp. 1 - 5, 2008.
[91] A. I. C. Golbreich, "Combining SWRL rules and OWL ontologies with Protégé OWL Plugin, Jess, and Racer," Université Rennes 1 France, 2004.
[92] P. P.-S. I. Horrocks, H. Boley, S. Tabet, B. Grosof, and M. Dean, "SWRL: A Semantic Web Rule Language Combining OWL and RuleML.," 2004.
[93] J. J. S. Zapater, "Ontologías para servicios web semánticos de información de tráfico: descripción y herramientas de explotación," Universitad de Valencia, España, 2006.
[94] P. Patel-Schne, "Safe Rules for OWL 1.1," Bell Labs Research, Alcatel-Lucent, 2008.
[95] "Overview of OWL-Safe Rules.," *Disponible en: http://code.google.com/p/owl1-1/wiki/SafeRulesOverview. Consultado el 26 de enero de 2012.*
[96] I. Horrocks., "Ontologies and the Semantic Web.," Oxford University Computing Laboratory, 2008.
[97] "The Rule Markup Initiative," *Disponible en: http://ruleml.org. Consultado el 26 de enero de 2012.*
[98] "Jess: the Rule Engine for the JavaTM Platform," *Disponible: http://www.jessrules.com/. Consultado el 27 de enero de 2012.*
[99] "Jess y su integración con JADE," *Disponible en: http://jess-jade.wetpaint.com/. Consultado el 27 de enero de 2012.*
[100] "Introduction and Features," *Disponible en: http://kaon2.semanticweb.org/. Consultado el 27 de enero de 2012.*
[101] M. Lama, E. Sánchez, "Proyecto Summa: Elearning multimodal y adaptativo. Análisis de técnicas de aprendizaje con ontologías," 2008.
[102] "Jena – A Semantic Web Framework for Java," *Disponible: http://jena.sourceforge.net/index.html. Consultado el 27 de enero de 2012.*
[103] "Jena 2 Inference support, The transitive reasoner," *disponible en: http://jena.sourceforge.net/inference/#transitive.*
[104] P. Barrera., "Gestión de la información multimedia en internet gestión del conocimiento DAML y ontologías consensuadas," 2003.
[105] Z. Zhang, "Ontology Query Languages for the Semantic Web," The University of Georgia, Athens, Georgia, 2005.
[106] P. J. H. Richard Fikes, and Ian Horrocks., "OWL-QL - a language for deductive query answering on the Semantic Web," *Journal in Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 2, 2004.
[107] J. G. a. D. B. Brian McBride, "RDF Test Cases," *W3C*, 2004.
[108] L. Dodds, "Introducing SPARQL: Querying the Semantic Web," *Disponible en: http://www.xml.com/pub/a/2005/11/16/introducing-sparql-querying-semantic-web-tutorial.html?page=1. Consultado el 28 de octubre de 2011.*, 2005.

[109] R. M. o. Volker Haarslev, and Michael Wessel, "Querying the Semantic Web with Racer + nRQL," in *In Proceedings of the KI-2004 International Workshop on ADL'04*, 2004.
[110] I. H. Franz Baader, Ulrike Sattler, "Description Logics for the Semantic Web," *KI - Künstliche*, 2001.
[111] Racer Systems GmbH & Co. KG, "RacerPro User's Guide Version 1.9.2," 2007.
[112] "SWRLTab Plugin," *Disponible en: http://protege.cim3.net/cgi-bin/wiki.pl?SWRLTab. Consultado el 27 de enero de 2012.*
[113] M. K. De Suparna, "Ontology-based Context Inference and Query for Mobile Devices," in *Centre for Communication Systems Research*, University of Surrey, Guildford, United Kingdom, 2008, pp. 1-5.
[114] e. a. S. Seneff, "Exploiting context information in spoken dialogue interaction with mobile devices," in *MIT Computer Science and Artificial Intelligence Laboratory*, 2007.
[115] S. Seneff, *et al.*, "Exploiting context information in spoken dialogue interaction with mobile devices," 2007.
[116] J. H. Vishnu S. Pendyala, "Performing Intelligent Mobile Searches in the Cloud using Semantic Technologies," in *IEEE International Conference on Granular Computing*, 2010, pp. 381-386.
[117] W. S. Gao Honghao, "A Design and Implementation of Search Engine for Mobile Devices Based Chinese Semantics and Reasoning," in *International Conference On Computer Design And Appliations (ICCDA 2010)*, 2010.
[118] A. Burton-Jones, *et al.*, "A semiotic metrics suite for assessing the quality of ontologies," *Data & Knowledge Engineering*, vol. 55, 2005, pp. 84-102.
[119] J. Brank, *et al.*, "A survey of ontology evaluation techniques," 2005.
[120] E. Ramos, *et al.*, "Esquema para evaluar ontologías únicas para un dominio de conocimiento," *Enlace*, vol. 6, 2009, pp. 57-71.
[121] M. P. Villalón, "Metodología neon aplicada a la representación del contexto," *Tesis de Maestría, Departamento de Inteligencia Artificial, Facultad de Informática, Universidad Politécnica de Madrid*, 2010.
[122] M. P. Villalon, "Red de ontologías para el Camino de Santiago," *Proyecto Fin de Carrera, Departamento de Inteligencia Artificial, Facultad de Informática, Universidad Politécnica de Madrid*, 2009.
[123] "W3C," *Disponible en www.w3.org/. Consultado el 10 de diciembre de 2011.*
[124] "Art & Architecture Thesaurus® Online," *Disponible en http://www.getty.edu/research/tools/vocabularies/aat/. Consultado el 12 de diciembre de 2011.*
[125] "Enciclopedia Libre Universal en Español," *Disponible en: http://enciclopedia.us.es/index.php/Enciclopedia_Libre_Universal_en_Espa%C3%B1ol. Consultado el 12 de diciembre de 2011.*
[126] J J. M. Faerna García-Bermejo and A. Gómez Cedillo, "Conceptos fundamentales de arte, Alianza Editorial, 1ra. Edición.," 2007.
[127] A. G.-P. M. C. Suárez-Figueroa, "NeOn Methodology: Scenarios for Building Networks of Ontologies," in *16th International Conference on Knowledge Engineering and Knowledge Management Knowledge Patterns*, Acitrezza, Catania, Italy, 2008.
[128] "Open-Source Large Vocabulary CSR Engine Julius," *Disponible en: http://julius.sourceforge.jp/en_index.php. Consultado el 8 de octubre de 2011. .*
[129] V. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics - Doklady 10*, vol. Vol. 10, 1966, pp. 707-710.
[130] D. M. Iaian McCowan, John Dines, Daniel Gatica-Perez, Mike Flynn, Pierre Wellner, and Hervé Bourlard, "On the Use of Information Retrieval Measures for Speech Recognition Evaluation," IDIAP, Idiap-RR, 2005.

[131] A. LEE, "The Julius book," *Disponible en: http://julius.sourceforge.jp/juliusbook/en/index.html, 2010, Consultado el 8 de Octubre de 2011.*
[132] C. M. University, "Voxforge Spanish Model Released," *Disponible en: http://cmusphinx.sourceforge.net/2010/08/voxforge-spanish-model-released/comment-page-1/. Consultado el 8 de Octubre de 2011.* .
[133] "How to write a recognition grammar for Julius," *Disponible en: http://julius.sourceforge.jp/en_index.php?q=en_grammar.html. Consultado el 8 de octubre de 2011.*
[134] "Google Voice ahora en Español.," *Disponible en: http://www.poderpda.com/investigacion-y-desarrollo/google-voice-search-ahora-en-espanol-latam/. Consultado el 10 de octubre de 2011.*
[135] B. K. Hans-Ulrich Krieger, Thierry Declerck, "A Hybrid Reasoning Architecture for Business Intelligence Applications," in *Eighth International Conference on Hybrid Intelligent Systems*, 2008.
[136] A.P.C. Silva, V.C.M. Borges and M.A.R. Dantas, "A Framework for Processes Submission and Monitoring from Mobile Devices to Grid Configurations Utilizing Resource Matching," Federal University of Santa Catarina (UFSC),, Florianopolis - SC, Brazil,2007.
[137] Y.-C. L. Yue-Shan Chang, Pei-Chun Shih, "AIR: Agent and ontology-based Information Retrieval architecture for mobile Grid," in *IEEE Asia-Pacific Services Computing Conference*, 2008.
[138] "Jena Ontology API," *Disponible en http://jena.sourceforge.net/ontology/. Consultado el 12 de noviembre de 2011.*
[139] D. B. Luis Polo, Emilio Rubiera, Sergio Fernández, "Descripción semántica del contexto:1.0," *Proyecto MORFEO*, 2007.
[140] M. Ortiz, "Introducción a las Lógicas Descriptivas Segunda Parte," *El lenguaje de las Lógicas Descriptivas, Newsletter, Logic and Computation Mexican Group*, 2008.
[141] "The new DIG interface standard (DIG 2.0)," *Disponible en http://dl.kr.org/dig/interface.html. Consultado el 1 de noviembre de 2011.*
[142] F. Arias, *et al.*, "Diseño y Desarrollo de Mecanismos de Razonamiento MultiAgente para la Negociación de Energía Eléctrica Utilizando JESS Y JADE," *Revista Avances en Sistemas e Informática, ISSN*, vol. 1657, 2006, pp. 51-56.
[143] F. Madou, *et al.*, "Sistemas Expertos en Evaluación de Calidad Java," ed: CONESCAPAN, 2009.
[144] J. Moskal and C. Matheus, "Detection of Suspicious Activity Using Different Rule Engines—Comparison of BaseVISor, Jena and Jess Rule Engines," *Rule Representation, Interchange and Reasoning on the Web*, 2008, pp. 73-80.
[145] G. K. Aimilia Magkanaraki, Ta Tuan Anh, Vassilis Christophides, Dimitris Plexousakis, "Ontology Storage and Querying," *TECHNICAL REPORT No 308*, vol. Foundation for Research and Technology Hellas, 2002.
[146] "Pellet FAQ: Single Page Version," *Disponible en: http://clarkparsia.com/pellet/faq/single-page/#different-results. Consultado el 3 de noviembre de 2011.*
[147] R. G. Norbert Weisharpenberg, and Agnes Voisard, "An Ontology-Based Approach to Personalized Situation-Aware Mobile Service Supply," *GeoInformatica*, 2006.
[148] "Using the Protégé-OWL Reasoner API," *Disponible en: http://protege.stanford.edu/plugins/owl/api/ReasonerAPIExamples.html. Consultado el 15 de febrero de 2012.*
[149] "SQWRLQueryAPI," *Disponible en: http://protege.cim3.net/cgi-bin/wiki.pl?SQWRLQueryAPI. Consultado el 15 de febrero de 2012.*

[150] "The Automatic Speech Recognition for Spanish Language Project," *Disponible en: http://www.speech.cs.cmu.edu/sphinx/models/hub4spanish_itesm/. Consultado el 10 de noviembre de 2011.*
[151] "SQLite," *Disponible en: http://www.sqlite.org/. Consultado el 15 de febrero de 2012.*