UNIVERSIDAD DE MURCIA

DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS

Plataforma inteligente de diseño para todos para control de teléfonos móviles mediante habla en lenguaje natural

D. Pedro José Vivancos Vicente

2016

UNIVERSIDAD DE MURCIA

DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS

TESIS DOCTORAL

Plataforma inteligente de diseño para todos para control de teléfonos móviles mediante habla en lenguaje natural

Pedro José Vivancos Vicente

2016

Directores

Dr. Rafael Valencia García
Dr. Jesualdo Tomás Fernández Bréis

Agradecimientos

Esta tesis sólo ha sido posible gracias al apoyo que he tenido,
laboral y emocional, de las personas que me rodean.
A Judith por estar siempre conmigo y darme el mejor regalo (Uriel).
A mis padres por su empeño en tener un hijo doctor.
A Rafa por el tiempo que ha dedicado a este trabajo.
A Jesualdo por su apoyo durante toda mi carrera investigadora.

ÍNDICES

ÍNDICE DE CONTENIDO

Capítulo I. Introducción 15

I.1. Motivación de esta tesis 16

I.2. Estructura de este documento 16

I.3. Breve introducción de las tareas desarrolladas 17

Capítulo II. Estado del Arte 21

II.1. Web Semántica 21

II.1.1. Antecedentes 22

II.1.2. Fundamentos de la Web Semántica 23

II.1.3. Arquitectura de la Web Semántica 24

II.2. Ontologías 26

II.2.1. Definición 26

II.2.2. Tipos de Ontologías 28

II.2.3. Elementos de una ontología 32

II.2.4. Estructura de la ontología 33

II.2.5. Lenguajes formales de Ontologías 33

II.2.6. Anotaciones OWL en las ontologías 38

II.2.7. API de acceso a las ontologías 38

II.3. Sistemas de Procesamiento de Lenguaje Natural 39

II.3.1. Definición 39

II.3.2. Antecedentes 40

II.3.3. Niveles del procesamiento del lenguaje natural 43

II.3.4. GATE 46

II.3.5. Procesamiento de lenguaje natural para el reconocimiento e interpretación de expresiones temporales 52

II.4. Reconocimiento de voz 58

II.4.1. Su uso en la industria 59

II.4.2. Estructura del habla 60

II.4.3. Procesamiento del habla 61

II.4.4. Modelos en el reconocimiento de voz 62

II.4.5. Algoritmos empleados en el reconocimiento de voz 64

II.4.6. Optimización de los sistemas de reconocimiento de voz 67

II.5. Cloud Computing 68

II.5.1. Características del cloudcomputing 68
II.5.2. Estudio de proveedores de cloudcomputing 69
II.5.3. Soporte de tecnologías para desarrollo basado en Java 76
II.6. Plataformas móviles 84
Capítulo III. Arquitectura propuesta 89
III.1. Introducción 89
III.1.1. Arquitectura funcional 89
III.1.2. Arquitectura cloud-computing 96
III.1.3. Despliegue en la nube 97
III.1.4. Ontología 100
III.2. Subsistema ASR 103
III.2.1. Tipo de reconocimiento de voz empleado 103
III.2.2. Evaluación de diferentes soluciones 104
III.2.3. Solución de reconocimiento de voz empleada 106
III.2.4. Consideraciones de la solución seleccionada 107
III.3. Subsistema NLP 108
III.3.1. Descripción general del subsistema 108
III.3.2. Recursos lingüísticos empleados 115
III.3.3. Interpretación de expresiones temporales 128
III.3.4. Resolución de ambigüedades 141
III.3.5. Resultado motor NLP: comando estructurado 151
III.4. Aplicación móvil 152
III.4.1. Introducción 152
III.4.2. Descripción general de la aplicación 153
III.4.3. Funciones de la aplicación 153
III.4.4. Requisitos específicos 155
III.4.5. Arquitectura software de la aplicación móvil 160
III.4.6. Diseño de la plataforma 163
Capítulo IV. Validación del sistema 171
IV.1. Introducción 171
IV.2. Pruebas en servidor ASR 173
IV.3. Validación por usuarios finales 175
IV.4. Sistema integrado 177
IV.4.1. Técnicas de Integración y pruebas de software 178

IV.4.2. Diseño y definición de la integración y validación de los componentes. ... 182
IV.4.3. Implementación de la integración de los componentes. ... 187
IV.4.4. Pruebas de rendimiento en la nube ... 190
IV.5. Conclusiones ... 192
Capítulo V. Conclusiones y líneas futuras ... 193
V.1. Conclusiones ... 193
V.2. Líneas futuras ... 198
Anexo I. Ejemplos de comandos XML ... 201
Anexo II. Comandos de voz implementados ... 209
Calendario de eventos ... 209
Comando AddCalendarEvent ... 209
Comando RemoveCalendarEvent ... 211
Comando ModifyCalendarEvent ... 212
Agenda de contactos ... 213
Comando AddContact ... 213
Comando RemoveContact ... 213
Comando ModifyContact ... 214
Sistema de alarma/despertador ... 215
Comando SetAlarmClock ... 215
Comando UnsetAlarmClock ... 216
Comando ModifyAlarmClock ... 216
Gestión de comandos no legibles ... 217
Comando NoActionRecognized ... 218
XML Schema empleado ... 218
Anexo III. Test socio-lingüístico ... 223
Referencias ... 231

ÍNDICE DE FIGURAS

Figura 1. Evolución de la Web a la Web Semántica (W3C Oficina España, 2007) *24*
Figura 2. Arquitectura en capas de la Web Semántica *24*
Figura 3. Ejemplo de grafo RDF *34*
Figura 4. Niveles del lenguaje OWL. *37*
Figura 5. Cuadro resumen de los niveles del lenguaje humano *43*
Figura 6. Modelo de Grafo de Anotaciones *48*
Figura 7. Regla JAPE para identificar direcciones IP *52*
Figura 8. Ejemplo de grabación de voz *59*
Figura 9. Fragmento de modelo lingüístico de en formato ARPA-LM para el portugués. *63*
Figura 10. Fórmula de cáculo del Word Error Rate (WER). *68*
Figura 11. Fórmula de cálculo de la precisión. *68*
Figura 12. Soporte de tecnologías para desarrollo basado en Java. *77*
Figura 13. Soporte al proceso de desarrollo y a la productividad el desarrollador. *79*
Figura 14. Características arquitectónicas. *81*
Figura 15. Decisiones de negocio. *84*
Figura 16. Comparativa de las principales plataformas móviles. *85*
Figura 17. Porcentaje de teléfonos inteligentes vendidos según su sistema operativo en el mundo (fuente: Gartner Group). *86*
Figura 18. Ventas en el mundo a usuarios finales de Smartphones por sistema operativo en 2013 (Millares de unidades) (fuente: Gartner Group) *87*
Figura 19. Diseño global del sistema propuesto. *91*
Figura 20. Arquitectura funcional del motor de NLP para el análisis de comandos. *93*
Figura 21. Arquitectura funcional servidor ASR. *95*
Figura 22. Arquitectura funcional cliente móvil. *96*
Figura 23. Arquitectura del servicio desplegado en la nube. *99*
Figura 24. Arquitectura desplegada en la nube. *100*
Figura 25. Ontología que representa el modelo del comando *101*
Figura 26. Ejemplo de detección de entidades a partir de los recursos desarrollados. *103*
Figura 27. Arquitectura del motor de procesamiento de lenguaje natural. *109*
Figura 28. Pipeline de GATE para el procesamiento de órdenes verbales. *110*
Figura 29. Ejemplo de pruebas con la aplicación gráfica del Framework GATE. *112*
Figura 30. Código Java de inicialización de GATE. *113*
Figura 31. Ejemplo de detección de entidades a partir de los recursos desarrollados *116*
Figura 32. Parte del fichero CREOLE con el plugin de GATE desarrollado. *117*

Figura 33. Parte del fichero tokenizer.rules que describe el VOCALI Tokenizer desarrollado. *120*
Figura 34. Parte del fichero lists.def desarrollado. *122*
Figura 35. Parte del fichero main.jape desarrollado. *124*
Figura 36. Parte del fichero general_persona.jape desarrollado. *127*
Figura 37. Parte del fichero general_telefono.jape desarrollado. *128*
Figura 38. Arquitectura del sistema de detección y resolución de expresiones temporales. *129*
Figura 39. Extracto del fichero timex_fechasfuzzy.lst. *133*
Figura 40 Parte del fichero general_fecha.jape desarrollado. *136*
Figura 41 Parte del fichero general_timesofday.jape desarrollado. *140*
Figura 42. Anotaciones temporales con información relativa sobre fechas y hora. *140*
Figura 43. Ejemplo de ambigüedad. *143*
Figura 44. Arquitectura del sistema de desambiguación. *147*
Figura 45. Ejemplo de ambigüedades donde una anotación está incluida dentro de otra. *147*
Figura 46. Ejemplo de resolución de ambigüedades. *148*
Figura 47. Ejemplo de ambigüedades donde existen anotaciones de tipo Action y Application en posiciones avanzadas del comando. *149*
Figura 48. Ejemplo de eliminación de anotaciones de tipo Action y Application. *149*
Figura 49. Ejemplo de ambigüedad de la entidad Application. *150*
Figura 50. Ejemplo de comando sin entidad de tipo Action. *151*
Figura 51. Arquitectura funcional cliente móvil *161*
Figura 52. Funcionamiento del módulo de conexión servicio *162*
Figura 53. Funcionamiento del módulo de ejecución de comandos *163*
Figura 54. Caso de uso de la aplicación terminal móvil. *164*
Figura 55. Interfaz móvil de la aplicación. *167*
Figura 56. Interfaz de respuesta con el comando XML *167*
Figura 57. Acción realizada en el terminal móvil *167*
Figura 58. Interfaz Android por defecto *168*
Figura 59. Interfaz con iconografía del plano. *169*
Figura 60. Interfaz en alto contraste. *170*
Figura 61. Diseño global del sistema final. *172*
Figura 62. Similitud de cadenas de texto basada en la distancia de Levenshtein. *174*
Figura 63. Ejemplo de cálculo de la distancia de Levenshtein. *175*
Figura 64. Resultados validación ASR. *175*
Figura 65. Estrategias de validación del sistema. *176*
Figura 66. Resultados validación ASR. *177*
Figura 67. Tipos de pruebas de software *180*
Figura 68. Tipos de pruebas de regresión. *181*
Figura 69. Tipos de pruebas de validación. *182*

Figura 70. Descomposición en componentes del sistema..... 183
Figura 71. Fragmento del comando AddCalendar..... 185
Figura 72. Orden del proceso de integración en el sistema. 186
Figura 73. Conexión segura entre dispositivo móvil y servidor ASR..... 188
Figura 74. Conexión segura entre dispositivo móvil y servidor ASR..... 189
Figura 75. Fichero XML Schema (XSD) empleado. 221

LISTA DE ACRÓNIMOS

Término	Significado
AC	Aprendizaje Computacional
ANNIE	A Neraly-New Information Extraction System
AIE	Adaptive Information Extraction
API	Application Programming Interface
ASR	Automatic Speech Recognition
BNF	Backus-Naur Form
CERN	European Organization for Nuclear Research
COHSE	Conceptual Open Hypermedia Service
CREAM	CREAtion of Metadata
CSS	Cascading Style Sheets
DAML	DARPA Agent Markup Language
DARPA	Defense Advanced Research Project Agency
DL	Description logic
DTD	Document Type Definition
FVP	Facet Value Pair
GATE	General Architecture for Text Engineering
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IA	Inteligencia Artificial
II	Information Integration
IDF	Inverse Document Frequency
IE	Information Extraction
IEEE	Institute of Electrical and Electronics Engineers
IR	Information Retrieval
IVR	Interactive Voice Response
KB	Knowledge base
KIMO	KIM Ontology
KMi	Knowledge Media Institute
KNN	K nearest neighbors
MUC	Message Understanding Conference
NER	Named-entity recognition
NLP	Natural language processing
OIL	Ontology Inference Layer
OWL	Web Ontology Language
PDF	Portable Document Format
PLN	Procesamiento del Lenguaje Natural
POS-Taggers	Part-of-speech Taggers
QL	Query Language
RDF	Resource Description Framework

RDFS	Resource Description Framework Schema
RNE	Reconocimiento de Nombres de Entidades
S-CREAM	Semi-automatic CREAtion of Metadata
SemTag	Semantic Tag
SGML	Standard Generalized Markup Language
SHOE	Simple HTML Ontology Extensions
TF	Term Frequency
TOR	Termino-ontologías
TTS	Text-To-Speech
TXL	Turing eXtender Language
URI	Uniform Resource Identifier
URN	Uniform Resource Name
W3C	World Wide Web Consortium
WWW	World Wide Web
XML	eXtensible Markup Language

Capítulo I. INTRODUCCIÓN

Las interfaces de control de los terminales móviles son complicadas de usar y no han evolucionado al mismo ritmo que las nuevas aplicaciones que han surgido a partir de él. Aunque la aparición de precisas pantallas táctiles y de nuevos sistemas operativos con mejores APIs y controles gráficos ha mejorado mucho el uso de estos terminales, ciertas operaciones siguen siendo tediosas y complicadas, más aún si la persona se encuentra realizando operaciones que no le dejan prestar toda la atención en el teléfono móvil. Por ejemplo, imagínese escribiendo un correo electrónico o un SMS mientras anda en un teclado virtual de una pantalla táctil. Incluso en un dispositivo con una interfaz táctil tan elaborada y precisa como la de un iPhone es muy complicado no equivocarse mientras se escribe incluso cuando la vista está fija en la pantalla del dispositivo.

Además, cada vez nos encontramos en situaciones donde necesitamos una respuesta rápida y eficaz del dispositivo en entornos donde incluso no podemos manipular el móvil manualmente, sin contar con personas que tengan una movilidad reducida (y que actualmente no pueden tener acceso a un smartphone por las interfaces que incorporan) o personas mayores, que ven las interfaces de estos dispositivos demasiado complejas de usar.

El uso de tecnologías de reconocimiento de voz, que convierten la voz en texto, y de tecnologías de Procesamiento de Lenguaje Natural y de representación de conocimiento, que son capaces de interpretar el significado de un texto, permitirán comunicarnos con nuestros dispositivos móviles y acceder a todas las aplicaciones que incorporan de una manera natural y sencilla, pues el uso de la voz es la interfaz común a la que estamos acostumbradas las personas, pues es nuestra forma de comunicarnos, mientras que otras interfaces como teclados, ratones, o interfaces táctiles son artificios que hemos inventado las personas para poder comunicarnos con máquinas incapaces de lidiar con la complejidad del lenguaje humano.

El objetivo principal de esta tesis radica en el desarrollo de una interfaz de diseño para todos, que permita el manejo y la interacción con un dispositivo móvil a través de ordenes en lenguaje natural. En resumen, el trabajo pretende desarrollar y mejorar una tecnología que permita a los sistemas informáticos “entender” el lenguaje coloquial, es decir, el lenguaje

usado por las personas para expresarse en su entorno. Estas tecnologías pretenden ser aplicadas en este trabajo en la interacción con los dispositivos móviles y toda la funcionalidad que permiten hoy día.

El resultado de esta tesis favorecerá la adaptación de estos dispositivos móviles a personas discapacitadas para que puedan utilizar toda la funcionalidad que proporcionan de manera sencilla y cómoda por este colectivo. Así, otro de los objetivos sociales es desarrollar tecnologías para la ayuda a este colectivo que les permita solventar las dificultades que tienen en el día a día y gracias de un módulo de reconocimiento del habla en lenguaje natural implantado en estos dispositivos puede resultar de una ayuda inestimable para conseguir una mayor integración de estos colectivos en la sociedad actual.

I.1. MOTIVACIÓN DE ESTA TESIS

Este trabajo ha perseguido desarrollar una tecnología que aborde los problemas planteados anteriormente en el apartado de introducción, es decir, el desarrollo y puesta en marcha de una interfaz basada en tecnologías semánticas y de procesamiento del lenguaje natural, que permita el manejo y la interacción con un dispositivo móvil a través de órdenes en lenguaje natural.

Este trabajo ha supuesto una serie de retos que afrontar y una gran cantidad de desarrollos para conseguir alcanzar los diferentes objetivos que se platean en este trabajo que se comenzaron cuando las plataformas móviles, es decir, los sistemas operativos empleados por los ahora tan extendidos *smartphones*, no incluían reconocimiento de voz y mucho menos asistentes inteligentes como Apple Siri, Google Now o Microsoft Cortana. En la actualidad este trabajo habría que plantearlo de una forma muy diferente, como se menciona en el apartado Capítulo V Conclusiones y líneas futuras

I.2. ESTRUCTURA DE ESTE DOCUMENTO

Para desempeñar esta labor, esta tesis comienza realizando una documentación del estado de la técnica de las tecnologías más adecuadas a aplicar en el trabajo y en el diseño de la arquitectura funcional para un sistema de reconocimiento de comandos de voz en lenguaje natural, como se puede ver en el Capítulo II de este documento, donde se hace especial hincapié en las tecnologías de Procesamiento de Lenguaje Natural (NLP) y de Web Semántica.

En el siguiente capítulo, denominado como "Arquitectura propuesta" se describen con detalle como se han emplado las tecnologías descritas en el estado del arte para obtener el sistema deseado. En este capítulo se habla de la ontología y recursos lingüísticos creados para la detección de comandos verbales en el dominio de los dispositivos móviles, del motor de reconocimiento e interpretación de expresiones temporales, o como se han resuelto las ambigüedades del lenguaje comunes en este tipo de sistemas mediante un sistema de inferencia, para poder ser capaces de procesar un comando completo en lenguaje natural y generar un comando estructurado siguiendo un esquema previamente definido.

Puesto que el sistema está pensado para aceptar órdenes de voz, otro de los componentes fundamentales del sistema consiste en un motor de reconocimiento de voz automático (ASR), que será el encargado de convertir la voz al texto que luego será procesado por todo el subsistema NLP.

Por último respecto a la arquitectura tecnológica, y puesto que este sistema se plantea originalmente para aplicaciones móviles para *smartphones*, se han desarrollado aplicaciones para plataformas móviles Blackberry y Android, así como toda la parte de comunicación entre las *apps* y los servicios de ASR y NLP que se han mencionado anteriormente.

Posteriormente se habla en el Capítulo IV del proceso de validación del sistema resultante de este trabajo para acabar con el Capítulo V con las conclusiones finales de este trabajo y futuras líneas de trabajo para continuar con el mismo.

I.3. Breve introducción de las tareas desarrolladas

Este trabajo se ha desarrollado dentro de un proyecto de investigación realizado por la empresa VÓCALI Sistemas Inteligentes, S.L., de la cual el autor es el Director de Estrategia en Innovación, y por tanto, como trabajo de investigación realizado con el fin de obtener una tecnología que dé lugar a productos comerciales, se ha abordado siguiendo las metodologías de desarrollo de la Ingeniería del Software apropiadas.

A modo de resumen, en este trabajo se han realizado diferentes acciones que han dado lugar al sistema final y que se resumen a continuación a modo de breve guía:

- Analizar detalladamente los resultados de investigación actual en el campo de las tecnologías de consulta en lenguaje natural y de extracción y representación del conocimiento.

- Desarrollo de entrevistas con potenciales clientes y colaboradores.
- Redacción y documentación de los requisitos funcionales y estructurales del proyecto.
- Diseñar y especificar la arquitectura funcional y modular del sistema a desarrollar.
- Generar las especificaciones técnicas para la fase de desarrollo.
- Diseñar un modelo ontológico para la representación del conocimiento dentro del dominio de la aplicación.
- Diseñar recursos lingüísticos para el procesamiento de órdenes vocales relacionadas con el control de aplicaciones del móvil.
- Buscar y adquirir bases de datos con nombres propios de personas, lugares y empresas que se utilizarán para la detección de entidades nombradas.
- Diseñar un motor de sistema de representación de expresiones temporales usando el estándar TIMEX2.
- Investigar e implementar un sistema capaz de resolver expresiones temporales complejas asociadas a horas del día.
- Investigar e implementar un sistema capaz de resolver expresiones temporales asociadas a fechas del calendario (días, días de la semana, meses, años,...)
- Investigar e implementar un sistema capaz de resolver expresiones temporales asociadas a intervalos de tiempo.
- Estudio de reglas para la detección de ambigüedades en comandos verbales para dispositivos móviles.
- Generación de un motor de inferencia para la detección y resolución de ambigüedades en función de las reglas obtenidas en tareas previas.
- Estudio de las fases NLP del sistema para su correcto funcionamiento.
- Creación de un "pipeline" con las diversas fases necesarias obtenidas en el estudio anterior.
- Módulo de generación de un comando estructurado a partir de los resultados obtenidos en tareas anteriores.
- Definición de un servicio de comunicación entre la aplicación servidor encargada de procesar un comando en lenguaje natural y la aplicación cliente que recibirá un comando estructurado con las operaciones a realizar en el terminal móvil.
- Implementación del Servicio Web para la comunicación entre ambas plataformas.

- Estudio y evaluación de las distintas plataformas móviles existentes.
- Diseño y definición de la aplicación móvil en las plataformas seleccionadas.
- Implementación de los subsistemas de las aplicaciones del terminal móvil a controlar.
- Desarrollo de subsistema de comunicación con los servidores web.
- Validación y pruebas.
- Diseño y definición de la integración y validación de todos los componentes.
- Implementación de la integración de los componentes.
- Estudio de los distintos proveedores de alojamiento de aplicaciones basadas en Cloud Computing.
- Adaptación y configuración de la aplicación en el servicio seleccionado.
- Pruebas de escalabilidad.
- Validación de las tecnologías desarrolladas por parte de personas discapacitadas.
- Generar la documentación asociada a esta tarea.

Capítulo II. ESTADO DEL ARTE

Dentro de este apartado se ha desarrollado un estudio exhaustivo de las tecnologías más importantes a desarrollar y utilizar en este trabajo de tesis. Más concretamente, la investigación se ha centrado en las tecnologías de Web Semántica, razonamiento, procesamiento del lenguaje natural y reconocimiento de voz.

II.1. WEB SEMÁNTICA

El concepto de "Web Semántica" fue acunado por el padre de la Web Actual, Tim Berners-Lee, junto con James Hendler y Ora Lassila en el referenciado artículo "The Semantic Web" (Berners-Lee et al., 2001).

Según los autores, la Web es un ente donde no existe significado que puedan explotar otros sistemas informáticos, sino que está pensada sólo para su uso por personas. El concepto de Web Semántica busca añadir contenido a la Web para que incluya "una semántica bien definida", de forma que la nueva Web no sea sólo legible por las personas sino también por agentes software, para conseguir automatizar ciertas tareas o explotar mejor la información que actualmente existe en la web.

Para explicar mejor el significado de esta Web Semántica, se hace uso de un breve extracto del artículo mencionado anteriormente:

"*La Web Semántica aportará estructura al contenido significativo de las páginas Web, creando un entorno donde los agentes software itinerantes de página en página, podrán llevar a cabo tareas sofisticadas para los usuarios […] la Web Semántica es una red de datos que puede ser procesada directa o indirectamente por máquinas. Es una Web extendida que permitirá a humanos y máquinas trabajar en cooperación mutua*". Según el artículo, l a Web Semántica debe contener características importantes que aseguren su interoperabilidad con las versiones actuales de la Web. Para ello, la Web Sematica se define como una versión extendida que dota de nuevas propiedades a la Web existente.

El concepto de Web Semántica ha sido también adoptado por el W3C (*World Wide Web Consortium)* que la define como "*una Web extendida, dotada de mayor significado, en la que cualquier usuario en Internet podrá encontrar respuestas a sus preguntas de forma más rápida y sencilla gracias a una información mejor definida*". Nuevamente se define la Web

Semántica como una extensión sobre la Web actual para dotar de inteligencia a la Web actual a partir de información estructurada. Para logar este objetivo, se necesita disponer de herramientas que faciliten la incorporación de descripciones explícitas, que permitan a los agentes software a procesar la información contenida en los sitios web y proporcionar a los internautas respuestas más precisas y de mayor calidad.

La Web es una estructura de grafo formado por multitud de nodos del mismo tipo que representan los contenidos interconectados por hiperenlaces. Estos contenidos se construyen mediante lenguajes de etiquetas para presentar la información. Lenguajes como HTML o XHTML no ofrecen información semántica del contenido o de los propios documentos, además de ser lenguajes muy limitados en expresividad. Puesto que la Web fue creada para ofrecer información a las personas, los lenguajes que emplean se limitan a definir cómo se presenta dicha información. Sin embargo, gracias a la Web Semántica se puede mejorar la expresividad mediante la incorporación de lenguajes estructurados que tienen como bae el formato XML (*eXtensible Markup Language*) y RDF (*Resource Description Framework*) que dotan a cada contenido de significado y de una estructura lógica, permitiendo mejorar el procesamiento de la información y la interoperabilidad entre los sistemas de información.

En resumen, Tim Berners-Lee reinventa la Web para solucionar las limitaciones que presenta la Web actual en lo que respecta al procesamiento automático de información, interoperabilidad con otros sistemas de información, o simplemente para manejar de manera más eficiente la gran cantidad de información y mejorar los procesos de búsqueda para encontrar de forma más rápida, precisa y sencilla.

II.1.1. ANTECEDENTES

Tim Berners-Lee concibió la Web para solucionar las limitaciones de intercambio de información entre investigadores durante su paso por el Consejo Europeo para la Energía Nuclear (CERN, http://home.web.cern.ch). Entonces, el padre de la WWW escribió una propuesta donde se planteaba una gran base de datos de texto enriquecido con enlaces a otros documentos que buscaba facilitar la forma de compartir y actualizar información.

Aunque esa propuesta no consiguió alcanzar el éxito entonces, gracias a la participación de Robert Cailliau. La versión revisada por Berners-Lee y Caillau unión las tecnologías de Internet,

y en concreto el estándar HTTP (*Hypertext Transfer Protocol*), y el HTML (*HyperText Markup Language*), lo que dio lugar a lo que ahora se denomina *World Wide Web* (http://info.cern.ch/hypertext/WWW/TheProject.html) y que supuso un un sistema global para la transferencia de información basado en identificadores únicos (hiperenlaces), y que resolvió la pérdida de información y permitió el intercambio de la misma entre los científicos del CERN. Los hiperenlaces se representan mediante el uso de direcciones URI (*Uniform Resource Identifier*) que son cadenas de caracteres que identifican de manera unívoca a un recurso dentro de una red. Estas URIs permiten acceder a otros recursos que en la web representan páginas, documentos o servicios.

Para poder poner todo en marcha, se creo el primer servidor Web y el primer navegador para poder mostrar la información. Este sistema concebido en el CERN fue utilizado inicialmente sólo en el CERN pero con el tiempo se extendió a otros centros expandiéndose de forma exponencial cuando se publicó la Web como un estándar abierto y formando el W3C para la regulación en su uso y crecimiento.

II.1.2. FUNDAMENTOS DE LA WEB SEMÁNTICA

Como se ha mencionado anteriormente, la Web Semántica plantea añadir información estructurada en la Web que defina el contenido de una forma que agentes software puedan "entender", y no sólo que la Web sea un repositorio de información y servicios sólo disponibles para personas.

Esto representa una evolución en la forma de representar los contenidos en la Web actual con respecto a la Web Semántica tal y como muestra la Figura 1, donde se muestra la estructura que plantean ambos tipos de Web: la tradicional y la semántica. En dicha imagen se puede apreciar como la Web actual tiene una estructura de grafo formado por nodos de la misma clase y enlaces unidireccionales, mientras que en la Web Semántica los nodos y enlaces representan relaciones explícitamente diferenciables fácilmente legibles tanto por personas como por aplicaciones software. Mientras que en la Web tradicional los nodos representan un documento de texto libre con formato sin ningún tipo de relación, en la Web Semántica los nodos no sólo contienen ese texto formateado, sino que incluyen información estructurada que representa un significado.

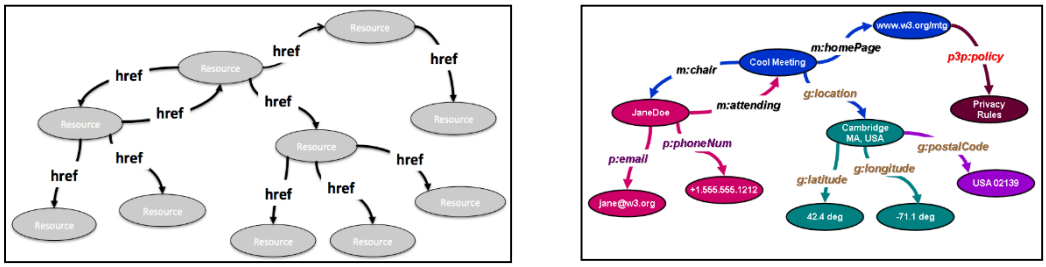


Figura 1. Evolución de la Web a la Web Semántica (W3C Oficina España, 2007)

Para conseguir representar ese "conocimiento" en forma de información estructurada, la Web Semántica requiere de elementos o herramientas y para ello se han elegido las ontologías como la forma actual más empleada de representación del conocimiento, ya que éstas ofrecen la capacidad de definir un vocabulario y describir relaciones entre los diferentes términos de manera flexible y sin ambigüedades, facilitando su interpretación por las máquinas y los humanos (Horrocks et al., 2003).

II.1.3. Arquitectura de la Web Semántica

Tim Berners-Lee definió en el año 2000 (http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html) la Web Semántica con una arquitectura capas (Figura 2) con el objetivo de proporcionar capacidades de procesamiento, razonamiento y deducción sobre los contenidos de la Web a los usuarios y agentes software.

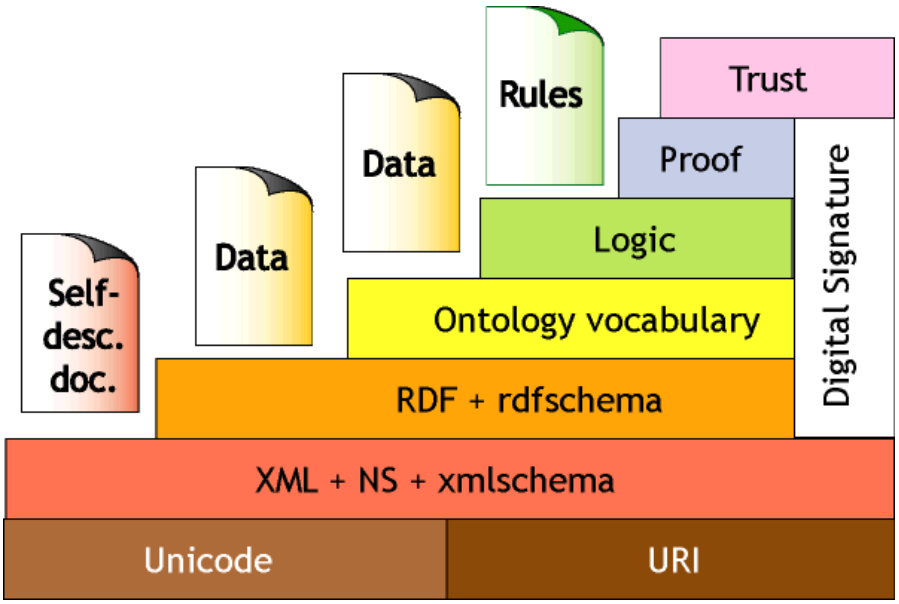


Figura 2. Arquitectura en capas de la Web Semántica

Cada capa tiene una función claramente definida dentro de la arquitectura como se comenta a continuación:

- **Unicode**. Esta capa permite que toda la información sea expresable en cualquier idioma dentro de la Web Semántica proporcionando una codificación estándar de caracteres que admite los distintos símbolos y alfabetos internacionales.
- **URI**. Ofrece un identificador uniforme único compuesta por una URL (*Uniform Resource Locator*), la cual describe cual es la localización del recurso, y una URN (*Uniform Resource Name*) que representa el espacio de nombres del mismo.
- **XML + Namespaces + XML Schema**. Esta capa sirve de base ofreciendo un formato común de documento para que los diferentes agentes software puedan interoperar unos con otros con independencia de las fuentes de información que se utilicen para crear los documentos. Esta capa, que está formada por varias tecnologías, cada cual con una función clara. Así, el lenguaje de marcado XML se erige como el formato común para el intercambio de documentos, con la posibilidad de realizar consultas gracias al lenguaje estándar de consultas XQuery. Otra de las tecnlogías empleadas es el espacio de nombres que se utiliza para cualificar atributos y elementos, y enlazarlos con los espacios de nombres identificados a través de referencias URI. Por último, se emplean plantillas *XML Schema* para facilitar la elaboración de documentos estándar.
- **RDF + RDF Schema**. RDF se basa en XML y lo mismo ocurre entre XML Schema y RDF Schema, y son las tecnologías que definen el lenguaje para representar el conocimiento dentro de la Web Semántica. Este lenguaje se forma mediante tripletas, mientras que el RDF Schema proporciona un vocabulario para describir las propiedades y las clases de los recursos RDF, con una semántica claramente definida con la que se pueden establecer jerarquías de generalización entre propiedades y clases.
- **Ontología**. Esta capa complementa la anterior, incluyendo más vocabulario que conseguir una descripción más precisa de conceptos, relaciones y propiedades con los que conceptualizar un dominio concreto.
- **Lógica**. En este caso se definen un conjunto de reglas de inferencia que los agentes software emplean para procesar y relacionar automáticamente la información a nivel semántico.
- **Prueba**. En esta capa se intercambian hechos y reglas estándares para facilitar la interoperabilidad entre recursos de la Web Semántica.

- **Firma digital**. Permite a los ordenadores y agentes software verificar la seguridad de la información adjuntada y que ésta se envió por una fuente confiable.
- **Confianza**. Tiene como objetivo la evaluación de las pruebas registradas en la capa "Prueba" para garantizar de forma exhaustiva que las fuentes de información son confiables.

De entre todas las tecnlogías mencionadas empleadas en la Web Semántica, cabe destacar las ontologías ya que constituyen el principal medio necesario para alcanzar el objetivo de esta nueva Web permitiendo definir formalmente las entidades y conceptos que se dan en diferentes dominios, así como las relaciones entre conceptos, destacando las relaciones jerárquicas ("IS-A"). Gracias a las ontologías se tiene una representación del conocimiento consensuada y reutilizable que puede ser compartida y utilizada automáticamente por cualquier agente software, como se describe más adelante en este capítulo.

II.2. ONTOLOGÍAS

II.2.1. DEFINICIÓN

Las ontologías tienen su origen en la Grecia antigua donde representan una ciencia que estudia la naturaleza de a existencia, aunque ha sido empleado en otros contextos en tiempos más modernos con significados diferentes.

En el contexto de la Inteligencia Artificial, sería Robert Neches quien definiera por primera vez en "Enabling technology for knowledge sharing" la palabra ontología, proporcionando la siguiente definición (Neches et al., 1991): "*Una ontología define los términos básicos y relaciones que conforman el vocabulario de un área específica, así como las reglas para combinar dichos términos y las relaciones para definir extensiones de vocabularios*", aunque la definición más extendida y popular fue proporcionada por Tom Gruber (1993): "*Una ontología es una especificación explicita de una conceptualización*". En esta definición, Gruber define ontología como una conceptualización está compuesta por objetos, conceptos y otras entidades que existen dentro de una determinada dominio, y las relaciones que pueden ser definidas entre ellos. Esta conceptualización se refiere a un modelo abstracto, que puede ser definido como una interpretación estructurada de una parte del mundo del que se identifican los conceptos más relevantes.

Sin embargo, la definición de Gruber ha recibido modificaciones por otros autores para hacerla más explícita, y cabe destacar la de Nicola Guarino, quien cuestionó la definión de Grube diciendo (Guarino, 1995): "*Un punto de inicio en este esfuerzo clarificador será el cuidadoso análisis de la interpretación dada por Tom Gruber. El problema principal de dicha interpretación es que se basa en la noción de conceptualización. Una conceptualización es un conjunto de relaciones extensionales que describen un estado particular, mientras que la noción que tenemos en mente es intencional, esto es, algo como una rejilla conceptual al que le imponemos varios posibles estados*", y proponiendo una nueva definición alternativa (Guarino, 1995): "*En el sentido filosófico, podemos referirnos a una ontología como un sistema particular de categorías que representa una cierta visión del mundo. Como tal, este sistema no depende de un lenguaje particular: la ontología de Aristóteles es siempre la misma, independientemente del lenguaje usado para describirla. Por otro lado, en su uso más típico en IA, una ontología es un artefacto ingenieril constituido por un vocabulario específico para describir una cierta realidad, más un conjunto de supuestos explícitos concernientes al significado pretendido de las palabras del vocabulario. Este conjunto de supuestos tiene generalmente la forma de teorías lógicas de primer orden, donde las palabras del vocabulario aparecen como predicados unarios o binarios, respectivamente llamados conceptos y relaciones. En el caso más simple, una ontología describe una jerarquía de conceptos relacionados por relaciones de subsunción: en los casos más sofisticados, se añaden axiomas para expresar otras relaciones entre conceptos y restringir la posible interpretación*"

Otro autor que redefinió el concepto de ontología de Gruber según su criterio fue Willem Nico Borst, quien incluyo el término "compartida" (Borst, 1997): "*Una ontología es una especificación formal de una conceptualización compartida*", donde el término "formal" hace referencia a la necesidad de disponer de ontologías comprensibles por las máquinas. Además, enfatiza la necesidad de consenso en la conceptualización, refiriéndose al tipo de conocimiento contenido en las ontologías, esto es, conocimiento consensuado y no privado de ahí el término "compartida".

Como último ejemplo de definición de ontología, o ampliación de la definición de Gruber, se destaca la que dió Rudi Studer, que complementa también la que facilitó Willem Nico Borst, y que definió la ontología como (Studer et al., 1998): "*Una ontología es una especificación formal y explícita de una conceptualización compartida*", y que es la definición que mejor se

adapta a las ontologías desde el punto de vista de este trabajo, destacando la siguiente ampliación que hizo Studer: "*Conceptualización se refiere a un modelo abstracto de algún fenómeno en el mundo a través de la identificación de los conceptos relevantes de dicho fenómeno. Explicita significa que el tipo de conceptos y restricciones usados se definen explícitamente. Formal representa el hecho de que la ontología debería ser entendible por una computadora. Compartida refleja la noción de que una ontología captura conocimiento consensual, esto es, que no es de un individuo, si no que es aceptado por un grupo*"

II.2.2. TIPOS DE ONTOLOGÍAS

En este apartado se han recogido las principales clasificaciones que se pueden encontrar en artículos científicos y técnicos sobre las ontologías, en función de diferentes aspectos, y que son las más extendidas.

II.2.2.1. Clasificación por el conocimiento que contienen

Esta clasificación tiene distintos puntos de vista según los autores a los que se haga referencia. En este trabajo se ha considerado interesante mencionar la de Mizoguchi y la de van Heijst.

En (Mizoguchi et al., 1995), Riichiro Mizoguchi y otros autores definen las siguientes categorías ontológicas:

- Ontologías del dominio: expresan conceptualizaciones específicas a un dominio en particular, describiendo los conceptos y sus relaciones respecto a un dominio específico.
- Ontologías de tarea: definen de forma específica el vocabulario de los conceptos de tal forma que el conocimiento del dominio es utilizable para realizar tareas específicas.
- Ontologías generales: ofrecen una descripción general sobre objetos, eventos, relaciones temporales, relaciones causales, modelos de comportamiento y funcionalidades.

Por otro lado, Gertjan van Heijst y otros autores hablan de una clasificación alternativa a la anterior que se centra en el volumen, tipo de estructura y en la conceptualización específica del conocimiento (van Heijst et al., 1997):

- Ontologías terminológicas o lingüísticas: donde se registran los términos que son usados para representar conocimiento en un dominio determinado, y que suelen emplearse para unificar el vocabulario de un dominio concreto.
- Ontologías de información: donde se representa la estructura de los registros de almacenamiento de una base de datos con el fin de proporcionar un marco estándar para el almacenamiento de la información.
- Ontologías para modelar conocimiento: se especifican conceptualizaciones de conocimiento. En este caso, exite una estructura interna más rica que las anteriores, y que las hace más atractivas para el desarrollo de sistemas basados en conocimiento. Estas ontologías se ajustan al uso particular del conocimiento que describen.

II.2.2.2. Clasificación por motivación

En este apartado se clasificación las ontologías según el motivo para el que se han elaborado.

Nuevamente existen diferentes clasificaciones según este aspecto según diferentes autores que se ha considerado merecedoras de comentar en este trabajo, comenzando por la que realiza G. Steve y sus colegas (Steve et al., 1997):

- Ontologías representacionales: son aquellas ontologías defindas para la representación del conocimiento y donde se especifican las conceptualizaciones que subyacen en los formalismos de representación de conocimiento (Davis et al., 1993), por lo que también se denominan meta-ontologías (también conocidas como *meta-level ontologie*s o *top-level ontologies*).
- Ontologías genéricas: que tienen como fin definir conceptos genéricos y fundacionales del conocimiento como relaciones PART-OF entre conceptos, la cuantificación, los procesos o los tipos de objetos. Estas ontologías se las suele llamar también abstractas o superteorías, ya que se emplean para representarr conceptos abstractos y pueden ser reutilizables en diferentes dominios.
- Ontologías del dominio: se emplean para representar el conocimiento especializado de un dominio o subdominio.

Nicola Guarino amplía la clasificación anterior añadiendo aquellas ontologías desarrolladas para actividades o tareas específicas, conocidas como *Task* Ontologies, en (Guarino, 1998) y que son:

- Ontologías genéricas o de alto nivel: describen conceptos generales que normalmente son independientes del dominio o problema particular. Se consideran, generalmente, de utilidad en todos los dominios o aplicaciones.
- Ontologías del dominio: expresan vocabulario específico de un dominio en particular, describiendo los conceptos y sus relaciones respecto a un dominio específico.
- Ontologías de tareas: especifican el vocabulario relacionado con una determinada actividad, tarea o artefacto definiendo los términos introducidos en la ontología de alto nivel.
- Ontologías de aplicación: recoge conceptos de un dominio y de una tarea en particular, los cuales son frecuentemente especializaciones de otras ontologías. Los conceptos aquí representados normalmente corresponden con funciones realizadas por las entidades de dominio mientras desempeñan una determinada actividad.
- Ontologías de grano grueso y de grano fino: en esta clasificación se etiqueta la ontología según la exactitud con la que reflejan la conceptualización. Para Guarino, según este criterio, existen dos tipos de ontologías: (1) las ontologías de grano grueso, y (2) las de grano fino. Las ontologías de grano grueso, también conocidas como ontologías *off-line*, consisten en un conjunto mínimo de axiomas escritos en un lenguaje de expresividad mínima, que desarrollan una axiomatización más rica, para apoyar un conjunto de servicios específicos a una amplia comunidad de usuarios que ya han acordado una conceptualización subyacente. Estas ontologías sólo son accesibles temporalmente por los usuarios para fines de referencia. Por otro lado, las ontologías de grano fino, llamadas también ontologías *on-line*, se desarrollan adoptando un dominio más rico y un conjunto más amplio de axiomas y relaciones conceptuales relevantes que permiten definir con bastante precisión los conceptos a los que se refiere. Estas ontologías proporcionan soporte al núcleo del sistema.

Roberto Poli propone una clasificación alternativa que distingue los siguientes tipos de ontologías (Poli, 2001):

- Ontologías generales: aquellas que describen las categorías superiores o fundamentales y oposiciones, además de sus conexiones de dependencia, y que se encuentran centradas en la arquitectónica de la teoría.
- Ontologías categóricas: las que estudian las diversas formas en las que una categoría registra los diversos estratos ontológicos, determinando la posible presencia de una teoría general que subsume sus concreciones.
- Ontologías del dominio: se refieren a la estructuración detallada de un contexto de análisis con respecto a los subdominios que lo componen.
- Ontologías genéricas: estas ontologías se encuentran ligadas a corpus lingüísticos y léxicos conceptuales que permiten la clasificación de los términos en varios niveles. Esto significa que cada término debería ser accesible, por defecto, únicamente en su sentido genérico, mientras que su significado especializado queda para cuando se activa una ontología del dominio específica.
- Ontología regional: se centran en analizar las categorías y sus conexiones de interdependencia para cada nivel ontológico.
- Ontología aplicada: son la aplicación concreta de la infraestructura ontológica a un objeto específico.

II.2.2.3. Clasificación por el grado de formalidad de la ontología

Otro criterio empleado en la clasificación de ontologías en función del grado de formalidad de la ontología de la misma, según afirma Poli en (Poli, 2003):

- Ontologías descriptivas: son aquellas que recogen los elementos del mundo, bien en general o bien de un dominio determinado. Poli defiendo que el mundo es el resultado de un complejo entramado de conexiones de dependencia y formas de independencia entre los elementos que lo componen. Estas ontologías descriptivas están pues relacionadas con la recopilación de información del mundo, entendiendo que en éste existen cosas materiales, plantas y animales, así como los productos de los talentos y actividades de animales y humanos, pensamientos, sensaciones y decisiones, así como el completo espectro de actividades mentales y reglas que se organizan en relación a la dependencia o independencia entre los mismos elementos.

- Ontologías formales: sonlas que tienen el propósito de filtrar y organizar los resultados de una ontología descriptiva en un dominio local o global. De esta forma, la formalidad de la ontología se base en la mereología, la topología y la teoría de la dependencia. Las interconexiones de las cosas, con objetos y propiedades, parte y todo que representan categorías puras que caracterizan aspectos y tipos de realidad son la base de una ontología forma.

II.2.3. ELEMENTOS DE UNA ONTOLOGÍA

Las ontologías representan una porción de conocimiento proporcionando un vocabulario común del dominio y las relaciones entre los distintos términos que forman dicho vocabulario de una manera formal.

Según Gruber (Gruber, 1993), las ontologías tienen los siguientes elementos:

- **Clases**. También conocidos como "conceptos", representan cualquier entidad que pueda ser identificada unívocamente que se pueda describir, poseer diferentes atributos y establecer relaciones con otros conceptos. Los conceptos de una ontología se suelen organizar en taxonomías, hasta el punto que en ocasiones una mera taxonomía se considera una ontologí (Studer et al., 1998).
- **Atributos**. Estos componentes forman parte de los conceptos de la ontología. En función de su origen, los atributos pueden ser específicos, pertenecen al propio concepto, o heredados, que vienen dados por las relaciones taxonómicas donde el conepto hijo hereda los atributos de la clase padre.
- **Relaciones**. Representan relaciones algebraicas entre conceptos de la ontología definiendo como están conectados diferentes conceptos del dominio.Formalmente se considera a una relación un subconjunto del producto cartesiano de *n* conjuntos. Estas relaciones son mayoritariamente entre dos conceptos y cada relación tiene un significado diferente (por ejemplo, relaciones taxonómicas IS-A o mereológicas PART-OF), que se pueden definir con distintas propiedades algebraicas como la simetría, reflexividad, transitividad, asimetría, etc.
- **Axiomas**. Modelan las "verdades" que siempre se cumplen en el modelo y en el dominio, por lo que definen expresiones que son siempre ciertas. Su utilidad en las ontologías puede ser definir el significado de los componentes ontológicos, definir

restricciones complejas sobre los valores de los atributos, argumentos de relaciones, etc. Estos axiomas verifican la corrección de la información especificada en la ontología y pueden ayudar en la generación de nuevo conocimiento.

- **Instancias**. Son las ocurrencias en el mundo real de los conceptos. Para que esto sea así, en una instancia todos los atributos del concepto tienen asignado un valor concreto.

II.2.4. ESTRUCTURA DE LA ONTOLOGÍA

Anteriormente se ha dicho que la definición adoptada en este trabajo de ontología del dominio es una especificación formal y explícita de una conceptualización compartida de un dominio. Una ontología define los términos, las relaciones existentes entre los conceptos, las reglas para poder combinar los términos para definir extensiones, etc. Esto supone unos beneficios que son los que han llevado a popularizar las ontologías en diversas áreas:

- Proporcionan un vocabulario comín para forma de representar y compartir el conocimiento.
- Ofrecen un formato formal de intercambio de conocimiento.
- Proporcionan un protocolo específico de comunicación.
- Facilitan una reutilización del conocimiento.

II.2.5. LENGUAJES FORMALES DE ONTOLOGÍAS

Los lenguajes formales para ontologías proporcionan la forma para representar el conocimiento en un formato intercambiable.

En este apartado se mencionan los lenguajes de ontologías más representativos que han surgido en los últimos años.

II.2.5.1. Simple HTML Ontology Extensions (SHOE)

El primer lenguaje ontológico se denominó SHOE (Luke et al., 1997) y consistía en una lenguaje de etiquetas para definir ontologías en la Web. La información de la ontología se introduce en un mismo archivo HTML que comúnmente se emplea en la Web. El lenguaje permitía definir conceptos, relaciones entre ellos y reglas de inferencia expresadas en forma de cláusulas de Horn (Heflin et al., 1998).

Sin embargo este lenguaje tiene una importante limitación por no disponer de ningún mecanismo para expresar negaciones o disyunciones. Actualmente la aparición de lenguajes

más potentes para representar ontologías han hecho que este lenguaje quede en desuso, aunque dispone de herramientas, API y editores de anotaciones, entre otros.

II.2.5.2. Resource Description Framework (RDF)

El lenguaje RDF representa los datos modelándolos como tripletascon el fin de representar recursos y las relaciones que pueden ser establecidas entre ellos. Cada tripleta está compuesta por dos nodos (sujeto y objeto) unidos por un arco (predicado) (Klyne & Carroll, 2004):

- Sujeto: identifica el recurso mediante una URI, entendiendo que un recurso RDF puede ser cualquier cosa en un modelo de datos (documento, usuario, producto, etc.).
- Predicado: representa una propiedad del sujeto y nuevamente cada una de estas propiedades o predicados se identifica con una URI única.
- Objeto: en RDF un objeto puede ser otro recurso (en tal caso tendrá una URI para identificarlo) o un literal (se representan por una cadena de caracteres u otro tipo XML) que especifica el valor del predicado para un sujeto. En términos de RDF, un literal puede contener marcado XML pero no es interpretado por RDF. El modelo RDF distingue a los literales de los recursos restringiendo a los literales ser sujeto o predicado en una declaración.

La Figura 3 muestra un ejemplo de tripleta en RDF:

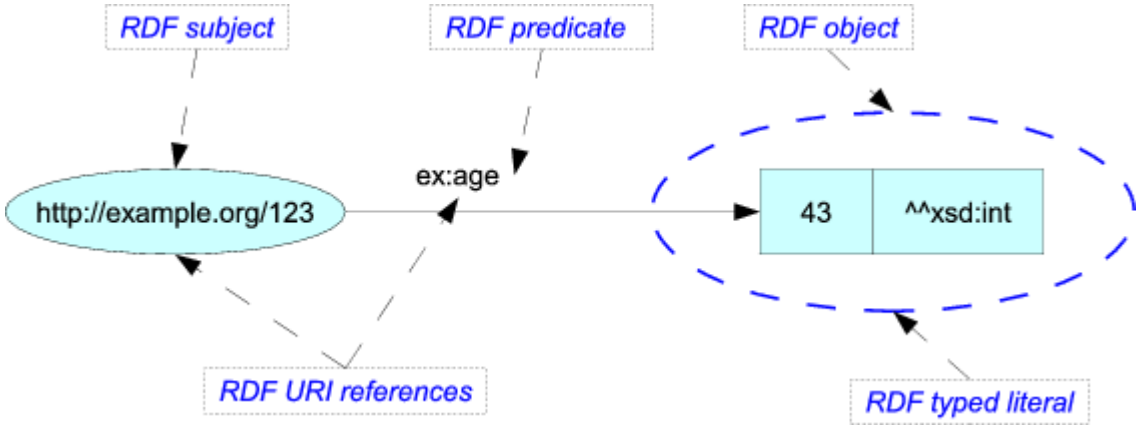


Figura 3. Ejemplo de grafo RDF

Como se puede ver en el ejemplo, cada recurso en RDF está descrito por un identificador único (URI) que es global. Las URIs son cadenas de caracteres que identifican al recurso, y está compuesta por un espacio de nombres y un identificador que debe ser único dentro de este

espacio de nombres. Dentro de las tripletas, los sujetos deben ser nodos identificados por una URI. Sin embargo, los objetos de las tripletas pueden ser tanto otros recursos como valores literales de propiedades.

II.2.5.3. Resource Description Framework Schema (RDFS)

Un esquema RDF (RDFS) supone una extensión del vocabulario básico de RDF (http://www.w3.org/TR/rdf-schema/) proporcionando un vocabulario para modelar datos RDF, constituyendo una extensión del vocabulario básico de RDF, y permitiendo expresar clases y relaciones jerárquicas, al igual que las propiedades que se asocian con clases (Cyganiak et al., 2004).

La combinación de RDFS con RDF es el lenguaje universal empleado para expresar el conocimiento en la Web Semántica. Esta combinación RDFS+RDF ofrece un lenguaje más potente que RDF para representar relaciones semánticas más complejas. Gracias a RDFS, RDF puede definir clases, relaciones de pertenencia entre clases, dominios y rangos para las propiedades, además de poder restringir en qué clases está cada propiedad y que valores pueden ser asignados. Este sistema es similar al de los tipos de los lenguajes de orientación a objetos pero en los lenguajes orientados a objetos se definen las clases en relación a las propiedades que se desea que posean las instancias, mientras que en RDFS es al contrario, las propiedades se definen basándose en las clases de recursos a los que éstas se pueden aplicar.

En cualquier caso, RDFS es un lenguaje ontológico simplista y su problema radica en que es demasiado primitivo y que carece de muchas características de modelado interesantes que otros lenguajes más actuales aportan, a pesar de que ofrece ciertas primitivas para el modelado de clases, relaciones de subclases, relaciones de subpropiedades y restricciones de dominio y rango, con un significado fijo. Entre las limitaciones existentes, se destacan las siguientes (Antoniou & Van Harmelen, 2004):

- La organización de vocabularios se limita a la definición de jerarquías.
- No se pueden representar algunas características de las propiedades. En concreto, no se puede declarar que una propiedad es transitiva, simétrica, inversa o única.
- No es posible expresar combinación booleana de clases.
- No permite establecer restricciones de cardinalidad.

- No permite expresar características de las propiedades tales como transitividad, simetría, unicidad, propiedad inversa, etc.

Sim embargo y a pesar de estas limitaciones, RDF+RDFS permite realizar tareas de razonamiento automáticas para inferir nuevas relaciones sobre una base de conocimiento dada.

II.2.5.4. Web Ontology Language (OWL)

OWL responde a la necesidad de extender los estándares y lenguajes ontológicos XML, RDF y RDFS para obtener una mayor capacidad de expresión, y que el lenguaje sea más fácil de entender y usar, estar formalmente especificado y ser capaz de proporcionar soporte a un razonamiento automático.

Antes de nacer OWL, se intentaron alcanzar estas condiciones mediante el desarrollo de extensiones, como DARPA Agent Markup Language (DAML) y Ontology Inference Layer (OIL). Este último permite realizar inferencias sobre el conocimiento gracias a descripciones lógicas que permiten realizar razonamiento. Viendo que ninguno de los dos conseguía cumplir completamente los requisitos deseados, se combinaron ambos lenguajes para crear DAML+OIL, pero no consiguió extenderse ya que estaban destinados a comunidades muy específicas como el Comercio Electrónico y Entornos científicos. De este modo se decide crear el Web Ontology Language (OWL) consiguiendo que se adapte a la arquitectura de la Web tradicional y a la Web Semántica.

Este lenguaje OWL se creó añadiendo mejoras imporantes respecto a los anteriores lenguajes como:

- Capacidad de ser distribuidas a través de varios sistemas.
- Escalable ante las necesidades de la Web
- Compatible con los estándares Web de accesibilidad e internacionalización.
- Abierto y extensible.

El encargado de crear y de mantener OWL fue el World Wide Web Consortium (W3C) y lo definen como *"un lenguaje de etiquetado semántico para publicar y compartir ontologías en la World Wide Web*". Este lenguaje se desarrolló como una extensión de RDFS, que emplea el modelo de tripletas de RDF, y deriva del lenguaje DAML+OIL. OWL permite una

interoperabilidad entre las ontologías y todo ello ha provocado que numerosos sistemas que usan DAML, OIL o DAML+OIL estén migrando a OWL.

Una de las curiosidades del lenguaje OWL es que realmente son 3 sublenguajes diseñados para ser usados por desarrolladores y usuarios específicos dependiendo del nivel de expresividad necesario. Cada sublenguaje ofrece diferencias en la tratabilidad computacional, como se comenta a continuación:

- **OWL Lite**: es el lenguaje más sencillo y se emplea cuando sólo es necesario clasificar en jerarquía los conceptos (clases) de la ontología y restricciones simples.
- **OWL DL** (Description Logic): es el lenguaje más empleado puesto que ofrece el máximo grado de expresividad, mientras se conservan completamente la computacionalidad (se garantiza que todas las conclusiones son computables) y resolubilidad (todas las computaciones terminarán en tiempo finito). Incluye todos los constructores del lenguaje OWL, pero solamente se pueden utilizar bajo ciertas restricciones.
- **OWL Full**: para aquellos casos donde no sólo se requiera de la máxima expresividad sino también de la libertad sintáctica de RDF sin garantías computacionales. Permite a una ontología aumentar el significado del vocabulario predefinido (RDF ó OWL).

Cada uno de estos sublenguajes es un subconjunto del siguiente, de forma que ambos pueden ser expresados uno dentro del siguiente y en los que pueden ser válidamente concluidos. La relación entre los niveles del lenguaje OWL se puede apreciar en la siguiente figura.

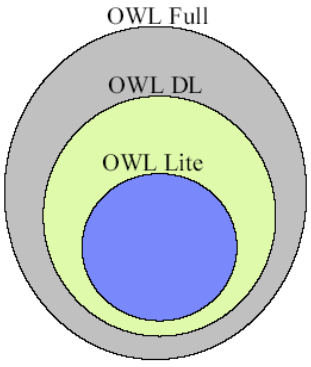


Figura 4. Niveles del lenguaje OWL.

El lenguaje utilizado para la construcción e integración de la ontología del dominio es OWL-DL.

II.2.6. ANOTACIONES OWL EN LAS ONTOLOGÍAS

OWL es el lenguaje ontológico empleado en el desarrollo de este trabajo. Dispone de sobrada capacidad para representar el vocabulario del dominio mediante los términos del mismo. Cada elemento de la ontología se etiqueta mediante anotaciones (Annotations). Estas anotaciones permiten además hacer que la ontología matenga vocabulario en diferentes idiomas.

Puesto que las anotaciones son un elemento muy importante de las ontologías definidas en OWL, se enumeran a continuación las diferentes anotaciones comunes a OWL:

- rdfs:comment: anota cada elemento mediante una descripción entendible por una persona. Un mismo elemento puede contener varias anotaciones de este tipo.
- rdfs:label: permite etiqueta utilizando lenguaje común el nombre o información de un recurso. Nuevamente, un mismo elemento de la ontología puede contener varias anotaciones de este tipo.
- preferredTerm: esta anotación no es estándar de OWL sino que se ha definido en este trabajo para representar qué término es el considerado más adecuado para repreentar un elemento ontológico y que sea el que se muestra al usuario en aquellos entornos que tengan interfaz. En este caso, un elemento sólo tendrá una anotación de este tipo.

Para que las ontologías empleadas sean multilingües se utilizará el atributo *lang* de las anotaciones, que permite definir el lenguaje en el que la anotación está escrita, podrán existir tantas anotaciones de un mismo término como idiomas en los que se encuentra escrito.

II.2.7. API DE ACCESO A LAS ONTOLOGÍAS

El uso de las ontologías en sistemas informáticos se ve facilitado si se puede trabajar en un nivel de abstracción superior al que ofrece el lenguaje ontológico empleado. Por ello existen APIs que permiten trabajar con diferentes lenguajes, y en concreto han proliferado varias APIs para trabajar con ontologías OWL. Entre ellas se destaca en este trabajo la OWLAPI (http://owlapi.sourceforge.net/index.html), por ser código abierto y libre.

OWLAPI es una biblioteca de código abierto basado en las licencias LGPL y Apache en Java para OWL que ofrece una API que entre otras cosas permite crear clases y métodos que específicamente pueden cargar y guardar archivos OWL. Esta API ha sido empleada en diferentes componentes y aplicaciones como editores de ontologías, herramientas de

anotación, consulta y agentes inteligentes. Dispone de varias interfaces de razonamiento que permiten explotar las funcionalidades de inferencia que el lenguaje semántico formal OWL proporciona como por ejemplo: el recorrido de la estructura ontológica,

La biblioteca OWL-API está compuesta por una serie de módulos que nombraremos a continuación:

- Una interfaz de programación para OWL 2.0.
- Un módulo para analizar y escribir documentos en RDF/XML.
- Un módulo para analizar y escribir documentos en OWL/XML.
- Un módulo para analizar y escribir documentos usando la sintaxis funcional de OWL.
- Un módulo para analizar y escribir documentos en Turtle.
- Un módulo para analizar documentos en el formato OBO.
- Un conjunto de interfaces con las que utilizar razonadores como HermiT, Pellet2, Fact++ o Racer.

El principal inconveniente de esta librería es que carga toda la estructura de la ontología en memoria para poder manipularla, por lo que conlleva un consumo de recursos excesivo en casos donde la ontología tiene un tamaño grande.

II.3. SISTEMAS DE PROCESAMIENTO DE LENGUAJE NATURAL

II.3.1. DEFINICIÓN

El lenguaje natural puede definirse como el medio oral o escrito que los humanos utilizan para propósitos generales de comunicación. El término "procesamiento del lenguaje natural" abarca un amplio conjunto de técnicas para la generación automática, manipulación y análisis de textos en lenguaje natural. Más concretamente, el procesamiento del lenguaje natural es un área de investigación y aplicación que explora cómo los ordenadores pueden ser usados para entender el significado del lenguaje que usamos los humanos para comunicarnos.

Liddy (2001) propone la siguiente definición: "*El procesamiento del lenguaje natural es un conjunto de técnicas computacionales teóricamente motivadas para analizar y representar naturalmente textos de origen natural en uno o más niveles de análisis lingüísticos para lograr el propósito de procesar el lenguaje humano por una serie de tareas o aplicaciones*".

Esta definición destaca la delicada labor que desempeñan estos sistemas, que necesitan de diferentes técnicas para llevar a cabo su misión. Además, resalta la capacidad de

representación de estos sistemas en diferentes niveles de análisis lingüísticos que serán analizados en los siguientes apartados.

Según Chowdhury (2003) el procesamiento del lenguaje natural es "*un área de investigación que explora cómo las computadoras pueden utilizarse para entender y manipular texto escrito en lenguaje natural o del habla para hacer operaciones útiles*".

Ambas definiciones establecen el objetivo del procesamiento del lenguaje natural, que se centra en el desarrollo de técnicas y construcción de herramientas que permitan a los sistemas informáticos entender y manipular los lenguajes naturales de tal manera que se facilite la comunicación entre seres humanos y sistemas informáticos.

II.3.2. ANTECEDENTES

La investigación en el campo del procesamiento del lenguaje natural tiene su comienzo hace varias décadas, a finales de 1940, donde la traducción automática fue la primera aplicación informática que relacionaba el mundo de los ordenadores con el lenguaje natural. Durante esta década y hasta finales de los años 50 se realizó un trabajo intenso sobre dos paradigmas fundamentales: los autómatas y los modelos probabilísticos. Esto supuso multitud de avances tecnológicos, comenzando por el primer intento de conseguir que un ordenador fuese capaz de procesar el lenguaje natural para realizar tareas de automatización de la traducciín entre los idiomas inglés y ruso (Locke & Booth, 1956). También en esta época se desarrolló el primer autómata según el modelo de computación algorítmica de Turing, que fue utilizado por McCulloch y Pitts (1943) como base para desarrollar, en 1943, un simplificado modelo de neuronas como tipo de elemento computacional descrito utilizando la lógica proposicional. Este modelo neuronal dio lugar al trabajo de autómatas finitos de Kleene (1951) y, posteriormente, a su trabajo en expresiones regulares (Shannon et al., 1956). En 1948, Shannon definió la teoría de autómatas (Shannon, 1948) y aplicó la teoría de la probabilidad de procesos de Markov para definir sistemas discretos, parecidos a los autómatas finitos, que procesaran el lenguaje humano. Chomsky (1956) consideró las máquinas de estados finitos para caracterizar las gramáticas y definió un lenguaje de estados como lenguaje generado por una gramática de estados finitos. Estos modelos iniciales llevaron a la teoría del lenguaje formal a incorporar el álgebra y la teoría de conjuntos para definir lenguajes formales como secuencias de símbolos. En este trabajo, Chomsky incluye la

renombrada jerarquía de gramáticas formales que generan lenguajes formales (Chomsky, 1956). Esta jerarquía se compone de cuatro niveles: (i) gramáticas de tipo 0, donde se incluyen todas las gramáticas formales; (ii) gramáticas de tipo 1, que recoge las gramáticas sensibles al contexto que generan los lenguajes sensibles al contexto; (iii) gramáticas de tipo 2, donde se sitúan las gramáticas libres de contexto que generan lenguajes independientes del contexto; y, por último, (iv) gramáticas de tipo 3, donde se sitúan las gramáticas regulares que generan los lenguajes regulares. Otro elemento fundamental desarrollado durante esta época fueron los algoritmos de probabilidad y de procesamiento del lenguaje, también aportación de Shannon (1948).

Entre los años 1950 y 1960, el procesamiento del lenguaje natural se dividió en dos paradigmas: simbólico y estocástico. El paradigma simbólico se originó a partir de las líneas de investigación de las obras de Chomsky y sus colegas sobre la teoría del lenguaje formal y la gramática generativa (Chomsky, 1956);(Moore, 1956), y de los inicios de Inteligencia Artificial, que comenzaron los investigadores John McCarthy, Marvin Minsky, Claude Shannon, y Nathaniel Rochester (McCarthy et al., 1955). Inicialmente los investigadores enfocaron su trabajo en el estudio de algoritmos estadísticos y estocásticos que incluían modelos probabilísticos y redes neuronales. Durante estos primeros años de investigación caben destacar los trabajos en razonamiento y lógica tipificado de Newell y Simon “Logic Theorist” (Newell & Simon, 1956) y “The General Problem Solver” (Simon et al., 1959). En estos primeros años destacan los sistemas simples de procesamiento de lenguaje natural que trabajaban en dominios concretos que utilizaban combinaciones de patrones y búsquedas basadas en palabras clave con simples heurísticas y técnicas de pregunta-respuesta.

El paradigma estocástico por otro lado fue investigado por departamentos de estadística e ingeniería eléctrica. El método bayesiano comenzó a aplicarse en los problemas de reconocimiento visual de caracteres a finales de los años 1950, como se mostraba en los trabajos de Bledsoe y Browning (Bledsoe & Browning, 1959), donde se presenta un sistema bayesiano de reconocimiento de texto. Otros trabajo a destacar es el de Mosteller y Wallace, (Mosteller & Wallace, 1964), donde se abordó el problema de la atribución de artículos mediante la aplicación de métodos bayesianos.

Durante los siguientes años de investigación se crearon más áreas de investigación gracias a los avances en tecnología y los nuevos conocimientos disponibles. El paradigma estocástico

desempeñó una labor muy importante en el desarrollo de algoritmos de reconocimiento de voz, como se verá más adelante cuando se hable de esta tecnología, particularmente con el uso de Modelos Ocultos de Markov y otros áreas como los teoremas desarrollados por Jelinek, Bahl, Mercer, y otros investigadores de IBM, junto con Baker de la Universidad Carnegie Mellon (Bahl & Mercer, 1976);(Baker, 1979);(Jelinek et al., 1975) sobre codificación de canal ruidoso y decodificación desarrollados.

El paradigma lógico desarrollado en Q-systems por Colmerauer y sus colegas sobre *metamorphosis grammars* (Colmerauer, 1975) donde se definió el primer formalismo gramatical basado en las cláusulas de Horn.

En cuanto al desarrollo de sistemas capaces de comprender el lenguaje natural, en el año 1972 apareció el sistema SHRDLU, que simulaba un robot integrado en un mundo de bloques que aceptaba comandos de texto en lenguaje natural (Winograd, 1972).

Por último, se empezaron a aplicar las técnicas en comprensión de lenguaje natural con la lógica de predicados para realizar representaciones semánticas, como se puede apreciar en el sistema LUNAR (Woods, 1973).

Entre los años 1980 y 1990 se volvieron a recuperar las ideas y trabajos que tuvieron éxito en los los años 1950 y 1960, a pesar de las críticas que recibieron entonces. En primer lugar se retomaron los modelos de estados finitos, gracias al uso de estos modelos el estudio de la fonología de estados finitos (Kaplan & Kay, 1981) y la morfología de los modelos de estados finitos de la sintaxis de Church (1980). Por otro lado, también se empezaron a utilizar métodos y enfoques probabilísticos de reconocimiento de voz y de procesamiento del lenguaje en lo que se denominó “retorno del empirismo”.

En estos últimos tiempos el área de Procesamiento de Lenguaje Natural ha sufrido importantes cambios. Actualmente se usan principalmente los modelos probabilísticos y los modelos *data-driven* en los métodos procesamiento del lenguaje natural. Hoy en día es muy común que las técnicas de análisis sintático y gramática, resolución de referencias y discurso han comenzado a incorporar modelos probabilísticos y a emplear metodologías de evaluación en el reconocimiento de voz y recuperación de la información. Además, gracias al incremento en la capacidad de los ordenadores, lo que ha permitido extender el uso del procesamiento del lenguaje natural en multitud de aplicaciones donde destacan aplicaciones de uso diario como son el reconocimiento de la voz y la detección de errores ortográficos y gramaticales, y

esto se ha incrementado gracias al auge de la Web, con el crecimiento exponencial de contenidos, ha destacado la necesidad de nuevos técnicas en los procesos de recuperación y extracción de información.

II.3.3. NIVELES DEL PROCESAMIENTO DEL LENGUAJE NATURAL

En este apartado se comentan los diferentes niveles de procesamiento de lenguaje natural según lo definieron Liddy y Feldman (1999) que destacaban la existencia de siete niveles interdependientes que las personas emplean para extraer significado de textos en lenguaje natural. La Figura 5 muestra un cuadro resumen que relaciona los diferentes niveles del lenguaje con cada una de las unidades lingüísticas que se procesan en esos niveles y, además, se incluyen las herramientas utilizadas en cada nivel.

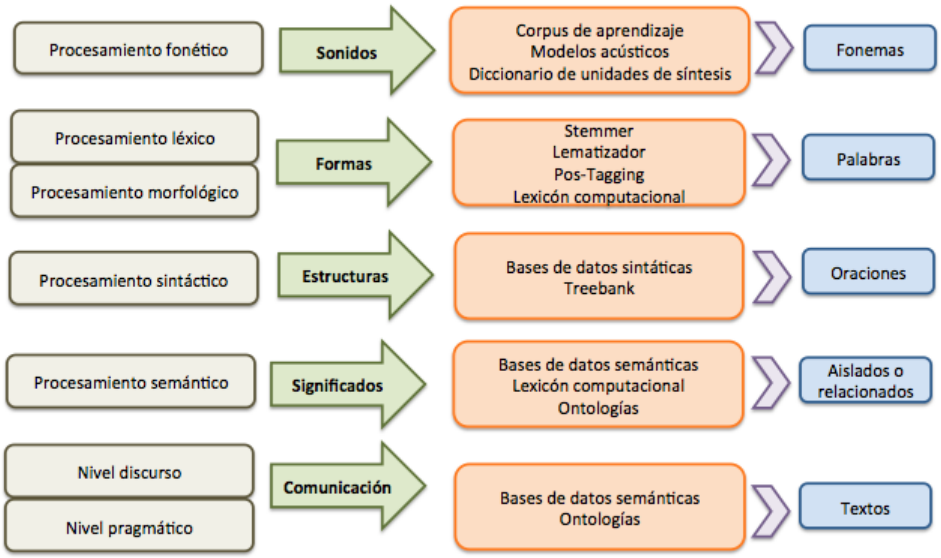


Figura 5. Cuadro resumen de los niveles del lenguaje humano

Las fases o niveles de procesamiento del lenguaje están interrlacionados entre si, de tal forma que durante el proceso de análisis el nivel actual requiere de información del nivel anterior. Un claro ejemplo se da cuando se trabaja en la desambiguación donde durante el procesamiento morfológico es necesario acceder a información generada por el procesamiento sintáctico para averiguar el papel que desempeña un término en una frase.

A continuación se enumeran los distintos niveles del lenguaje realizando una breve explicación de las diferentes técnicas de cad nivel.

II.3.3.1. Nivel fonético

Se emplea en los sistemas de reconocimiento de voz y en este nivel se descrbien las dimensiones físico-acústicas, articulatorias y auditivas de los sonidos en el lenguaje.

En este punto existen tres tipos de reglas que se utilizan para realizar el análisis fonológico: (i) reglas fonéticas, para analizar los sonidos producidos cuando se pronuncia una palabra; (ii) reglas fonológicas, para detectar las variaciones de pronunciación producidas cuando las palabras se hablan unidas; y (iii) reglas prosódicas, que permiten analizar la fluctuación del acento y la entonación cuando se pronuncia una frase.

En los sistemas de reconocimiento de voz, este nivel realiza el procesamiento fonológico para analizar y codificar las ondas producidas en una señal digitalizada para interpretar varias reglas o para establecer comparaciones con otro modelo de lenguaje.

II.3.3.2. Nivel morfológico

La morfología es "la rama de la lingüística que estudia la estructura interna de las palabras para delimitar, definir y clasificar sus unidades, las clases de palabras a las que da lugar (morfología flexiva) y la formación de nuevas palabras (morfología léxica).

Según esta definición, el nivel morfológico emplea *stemmers*, lematizadores o POS-Taggers para transformar una secuencia de caracteres en una secuencia de morfemas. Gracias a estos métodos se deteriman aspectos de cada palabra como tiempo, género, número, grado, etc. y se detectan sufijos, prefijos, sinónimos, generalizaciones, o especializaciones. Gracias a estas técnicas también se pueden clasificar las palabras (p. ej., sustantivo, verbo, adjetivo, adverbio, etc.). Las personas pueden descomponer una nueva palabra en sus morfemas constituyentes para entender el significado, que es lo que los sistemas de procesamiento morfológico intentan simular al reconocer el significado de cada morfema para obtener el significado total de la palabra.

II.3.3.3. Nivel léxico

A este nivel se busca obtener el significado de cada palabra de manera individual. Para ello es necesario que los sistemas de procesamiento del lenguaje natural dispongan de técnicas e información para conocer las categorías léxicas existentes en el lenguaje.

El análisis léxico de una palabra puede realizarse empleando diversas técnicas, y la más común es se basa en la asignación de etiquetas individuales a cada palabra del texto. Las

etiquetas determinan la categoría léxica de la palabra y, cuando una palabra puede tener más de una función dentro de una oración, se etiqueta con la categoría más probable en función del contexto.

II.3.3.4. Nivel sintáctico

Con la sintaxis se busca determinar las relaciones existentes entre las palabras dentro de una misma frase, y, en un nivel más general, las reglas que rigen estas oraciones con el fin de descubrir la estructura gramatical de la frase y analizar cómo las palabras se mezclan para componer oraciones gramaticalmente correctas.

En este nivel se emplean bases de datos con los patrones sintácticos más frecuentes del idioma que se esté analizando y analizadores sintácticos capaces de obtener la estructura gramatical de la secuencia de unidades léxicas para representarla en forma de árbol o red, dando como resultado una representación de la estructura de la oración que pone de manifiesto las relaciones de dependencia estructural de las palabras.

II.3.3.5. Nivel semántico

En este nivel se estudia el significado del lenguaje. El procesamiento semántico busca determinar los posibles significados de una frase a través de los significados de cada palabra y las relaciones entre ellas.

En este nivel son comunes las técnicas de desambiguación semántica, ya que es común que las palabras tengan varios significados. La desambiguación semántica se utiliza para seleccionar el signficado correcto en una frase de una palabra polisémtica. El típico ejemplo empleado al explicar este nivel es la palabra “banco”, que apareciendo siempre como un sustantivo, tiene muchos significados (sucursal bancaria, lugar donde sentarse,…)

Los métodos empleados para desambiguar suelen emplear información relacionada con la frecuencia en que aparece cada término en un corpus de entrenamiento, o bien utilizan el conocimiento pragmático del dominio del documento.

II.3.3.6. Nivel contextual

En el nivel del discurso o del contexto se procesan textos completos (o fragmentos de éstos) a diferencia de los niveles sintácticos y semánticos donde se trabaja a nivel de oración.

A este nivel un sistema de procesamiento de lenguaje natural analiza las propiedades del texto en su conjunto, entendiendo que las diferentes frases que forman el texto conectadas entre si forman un signficado más completo que las oraciones sueltas, es decir, que a diferencia del nivel semántico. En este nivel se usa la estructura semántica del nivel anterior para desarrollar una interpretación final de la oración en función del contexto del texto procesado.

En este nivel las técnicas más empleadas en este nivel son la resolución anafórica y el reconocimiento de la estructura del texto/discurso. La primera se basa en reemplazar palabras vacías semánticamente, como pronombres, por la entidad a la que referencian. Mientras que el reconocimiento de la estructura del texto/discurso determina las funciones de las sentencias en el texto y añade esta representación significativa al texto.

II.3.3.7. Nivel pragmático

En el nivel pragmático se emplean estrategias comunicativas enmarcándolas en un contexto socio-cultural.

Se trata por supuesto de uno de los niveles de análisis más complejos, puesto que el análisis que se realiza en este nivel se encuentra relacionado con los factores extralingüísticos que condicionan el uso del lenguaje en situaciones comunicativas concretas, factores que no pueden ser analizados por los niveles anteriores, como la intención comunicativa, el contexto verbal, y la situación o conocimiento del mundo.

En este análisis se utiliza el contexto para comprender el signficado del texto, por encima de los contenidos. Para realizar este procesamiento, los sistemas de análisis pragmático disponen de bases de conocimiento y módulos de inferencia que permiten interpretar las intenciones, los planes y los objetivos de un texto.

II.3.4. GATE

El módulo de procesamiento de lenguaje natural (NLP) es el más complejo e importante del trabajo realizado.

GATE (A General Architecture for Text Engineering) es una plataforma para el desarrollo de componentes de software para procesamiento de lenguaje natural.

La arquitectura explota el desarrollo de software basado en componentes, la orientación a objetos y el código portable. El marco de trabajo y el entorno de desarrollo están desarrollados en Java y disponibles como software libre bajo licencia de la librería GNU.

GATE ha sido desarrollada en la Universidad de Sheffield desde 1995 y ha sido utilizada en una amplia variedad de proyectos de investigación y desarrollo. La versión 1 de GATE, lanzada en 1996, fue utilizada en una amplia gama de contextos de análisis de lenguaje incluyendo Extracción de Información en: Inglés, Griego, Español, Sueco, Alemán, Italiano y Francés. Actualmente está disponible la versión 8 del sistema, una completa re-implementación y extensión de la versión original.

GATE como arquitectura, sugiere que los elementos de los sistemas de software que procesan lenguaje natural se pueden dividir en varios tipos denominados recursos. Estos recursos son partes reusables con interfaces bien definidas, y son una forma de arquitectura popular utilizada por ejemplo en Java Beans y Microsoft .Net. Los componentes de GATE son tipos específicos de Java Bean y hay tres tipos:

- *Recursos del Lenguaje (LR):* representa entidades como léxicos, recopiladores u ontologías.
- *Recursos de Procesamiento (PR):* representa entidades principalmente algorítmicas, como parsers, etiquetadores, lematizadores, traductores, reconocedores de voz, etc. En general, los PR incluyen a los LR, por ejemplo, un etiquetador usualmente incluye un léxico.
- *Recursos Visuales:* representa componentes de visualización y edición que participan en la GUI.

Todos los recursos de GATE son Java Beans, el modelo de plataforma en Java de componentes de software. Los Beans son simplemente clases en Java que obedecen ciertas convenciones en las interfaces. Estas convenciones permiten desarrollar herramientas como GATE para manipular componentes de software sin necesitar mucho conocimiento de ellos.

El conjunto de recursos integrados en GATE es conocido como CREOLE (Collection of Reusable Objects for Language Engineering). Todos los recursos existen como paquetes en archivos Java (archivos JAR) además de algunos datos de configuración en archivos XML. Cuando se ha desarrollado un conjunto apropiado de recursos, éstos pueden ser embebidos en la aplicación cliente objetivo utilizando el marco de trabajo de GATE.

Los documentos en GATE, recopilaciones y anotaciones son guardados en bases de datos de diferentes clases, visualizados en el entorno de desarrollo, y accedidos a nivel de código vía el marco de trabajo. Los recursos pueden ser cargados en GATE, y guardados en conjunto en un Data Store. Además, los recursos de procesamiento se pueden "unir" por medio de una tubería (pipeline), igual que en las tuberías de Unix donde la salida de una aplicación es la entrada a la siguiente. En GATE esto se denomina aplicación. GATE soporta documentos en una gran variedad de formatos incluyendo XML, RTF, email, HTML, SGML y texto plano. En todos los casos el formato se analiza y se convierte en un modelo unificado de anotaciones. Una anotación es un metadato unido a una sección particular del contenido del documento. La conexión entre una anotación y el contenido al que se refiere se hace por medio de dos indicadores que representen las localizaciones de inicio y del final del contenido del texto cubierto por el análisis. Una anotación debe también tener un tipo (o un nombre) que se utiliza para crear clases de anotaciones similares, ligado generalmente junto con su semántica.

Un documento en GATE puede tener unas o más capas de anotaciones, una anónima (también llamada *default*), y tantas *conocidas* como se requiera. Una capa de anotación se organiza como un gráfico dirigido acíclico (ver siguiente figura), en el cual los nodos tienen una localización determinada en el documento y los arcos representan las anotaciones que alcanzan la localización indicada por el nodo del inicio al de fin. Gracias a esta estructura, un conjunto de anotaciones (*Annotations Set*) mantiene un número de anotaciones y una serie de índices que proporcionan un acceso rápido a las anotaciones contenidas.

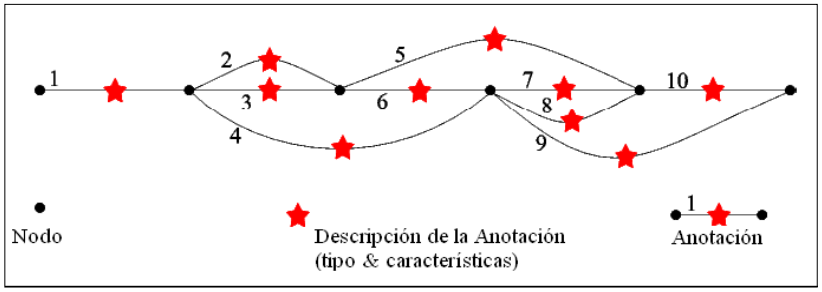


Figura 6. Modelo de Grafo de Anotaciones

II.3.4.1. ANNIE

Según algunos autores, las técnicas de extracción de información procesan un documento para identificar entidades y sus relaciones y luego formar una plantilla con la información identificada. ANNIE (A Nearly-New Information Extraction System) es un sistema aportado

por GATE. Originalmente fue desarrollado en el contexto de extracción de información. Una gran variedad de sistemas de este tipo han sido creados en varios lenguajes utilizando los componentes que fueron distribuidos con ANNIE. Es un sistema altamente modular, cuyos componentes se organizan en una arquitectura de estilo tubería. A continuación vamos a describir algunos de sus componentes más importantes:

- *Tokenizer:* Separa el texto en tokens simples como pueden ser números, símbolos de puntuación, y palabras.
- *Gazetteer:* Utiliza una serie de listas de palabras. Cada lista representa un conjunto de nombres, se utiliza un archivo índice para acceder a estas listas. Se explicará con más detalle en la siguiente sección.
- *Sentence Splitter:* Es una cascada de transductores de estado finito que separan el texto en sentencias.
- *Part of Speech Tagger:* Produce una etiqueta como una anotación en cada palabra o símbolo. Utiliza un conjunto de reglas y un léxico por defecto (resultado de entrenar en un gran corpus tomado del Wall Street Journal), ambos se pueden modificar si es necesario.
- *Semantic Tagger:* Basado en el lenguaje JAPE que describiremos más adelante. Contiene reglas que actúan en anotaciones asignadas a frases previamente, y produce como salida entidades de anotaciones.
- *Orthographic Coreference (NameMatcher):* Agrega identidad a los nombres de entidades encontrados por el etiquetador semántico.

La idea es crear una tubería con estos módulos para que se ejecuten en secuencia y cada uno realice el procesamiento correspondiente. Tras algunos pasos básicos como el etiquetado de partes de discurso y las búsquedas en listas mediante el módulo Gazetteer, se ejecuta el etiquetador semántico. El etiquetador semántico utiliza reglas que explotan las anotaciones realizadas en los pasos previos para producir anotaciones de entidades.

ANNIE se basa en el transductor JAPE y puede ser fácilmente extensible para agregar nuevos diccionarios y reglas. Además de los recursos de procesamiento incluidos en el sistema ANNIE, GATE provee una amplia variedad de recursos que, si bien no se cargan por defecto, se pueden configurar para que sí lo hagan.

II.3.4.2. Listas con GATE. Gazetteers.

La traducción de *gazetteers* es diccionario geográfico y proporciona un acercamiento inicial al significado de este concepto. Un diccionario geográfico contiene usualmente un listado alfabético de países con información estadística relativa a cada uno de ellos y otros diccionarios geográficos que listan información relativa a ciudades, pueblos, etc.

Una traducción más apropiada para *gazetteer* podría ser Tesauro. Un tesauro es una lista estructurada de descriptores o términos propios de un ámbito científico determinado, entre los cuales se establecen una serie de relaciones jerárquicas y asociativas. Además de la presentación alfabética, ofrecen una representación gráfica de las relaciones entre los descriptores.

En el ámbito de GATE los *gazetteers* consisten en un conjunto de listas que contienen nombres de entidades, tales como días de la semana, nombres de persona, ciudades, etc. que no tienen porque estar ordenados alfabéticamente. Su principal uso es ayudar en tareas de reconocimiento de nombre de entidades, aunque pueden ser usadas para otros propósitos.

En GATE cuando el *gazetteer* se ejecuta sobre un documento se crean anotaciones del tipo *Lookup* para cada secuencia coincidente en el texto. El *gazetteer* es independiente de cualquier *token* u otra anotación. Es decir, una entrada podría involucrar a más de un *token*. Una anotación *Lookup* será creada solamente cuando exista coincidencia con la entrada completa, las entradas parciales no serán coincidentes. Estas listas se crean en ficheros de texto plano en las cuales cada línea corresponde a un término, también se crea un fichero con extensión .def que hace de índice a todos estos ficheros de listas.

Es importante, sobre todo, cuando lo que se quiere procesar es lenguaje español, con tildes y eñes, estar seguro de que en el editor que se esté utilizando para crear estas listas se guarde el fichero en UTF-8.

Para facilitar este proceso el framework de GATE proporciona un editor Unicode, al que se puede acceder desde la barra principal: Tools > Unicode Editor. También proporciona las tareas básicas de un editor tradicional y al guardar pregunta el formato de codificación que se desea usar.

II.3.4.3. Reglas con GATE

GATE proporciona un conjunto de facilidades como JAPE (*Java Annotation Patterns Engine*). JAPE permite reconocer expresiones regulares de anotaciones en documentos. Es necesario destacar que un lenguaje regular sólo es capaz de describir conjuntos de cadenas de texto, sin embargo, el modelo de anotaciones de GATE está basado en grafos. El resultado es que en ciertos casos el proceso que ajusta es no determinístico (esto es, los resultados son dependientes de factores aleatorios como las direcciones en que los datos son guardados en la máquina virtual). Cuando hay estructuras en el grafo que necesitan algo más que una automatización regular para ser reconocidas, JAPE elige una alternativa arbitrariamente. En la práctica, en muchos casos los datos guardados en los grafos de anotaciones de GATE son secuencias simples, y pueden ser ajustadas determinísticamente mediante expresiones regulares.

Una gramática de JAPE consiste en un conjunto de frases, cada una de las cuales consiste en un conjunto de reglas patrón / acción. La parte izquierda (LHS) de las reglas consiste en un patrón de anotación el cual puede contener operadores de expresiones regulares (*, ?, +). La parte derecha (RHS) consiste en sentencias de manipulación de anotaciones. Anotaciones que se ajustan a la parte izquierda de una regla pueden ser referenciadas en la parte derecha por medio de etiquetas que se adjuntan a los patrones.

Las reglas de gramática pueden ser esencialmente de dos tipos. El primer tipo de reglas no involucra búsqueda en diccionarios, pero éstas pueden ser definidas utilizando un conjunto reducido de formatos posibles. En general, son directas y ofrecen poco potencial para la ambigüedad. El segundo tipo de reglas tiene un uso más pesado de diccionarios, y cubre un mayor rango de posibilidades. Esto no sólo significa que se van a necesitar muchas reglas para describir todas las situaciones, sino que además hay mucho más potencial para la ambigüedad. Esto lleva a la necesidad de ordenar y priorizar las reglas.

Por ejemplo, una única regla es suficiente para identificar direcciones IP:

```
Rule: IPAddress (
{Token.kind == number}
{Token.string == "."}
{Token.kind == number}
{Token.string == "."}
{Token.kind == number}
```

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

Figura 7. Regla JAPE para identificar direcciones IP

En cambio, para identificar una fecha, puede haber muchas variaciones posibles, por lo que se necesitan varias reglas. Esto también significa que hay un gran potencial para la ambigüedad. Por ejemplo, la palabra “Friday” puede ser el nombre de una persona o el día de la semana. Esto significa que será necesario mantener información sobre el contexto para eliminar la ambigüedad, o se podría adivinar a que se refiere la palabra basándose en su frecuencia de aparición.

El contexto se utiliza para definir bajo que situaciones se aplica un patrón. Por ejemplo, en el reconocimiento de una fecha, puede definirse un contexto en el que la fecha se reconozca sólo si el año está precedido por las palabras “in” o “by”. La parte derecha de una regla en JAPE puede contener código en lenguaje Java. Esto es útil para borrar anotaciones temporales y para manipular características de anotaciones previas.

Las gramáticas JAPE se escriben en archivos con extensión “.jape”, estos son parseados y compilados en tiempo de ejecución para luego ser ejecutados en el documento de GATE.

GATE provee una herramienta denominada “Jape Debugger” la cual permite encontrar errores en programas JAPE permitiendo al usuario ver en detalle cómo trabaja una regla en particular cuando se aplica a un rango del texto. La herramienta permite ver qué reglas fueron aplicadas y cuáles no, y además, por qué algunas reglas fueron o no aplicadas. Así mismo, es posible colocar puntos de corte para reglas particulares permitiendo al usuario ver cómo se aplicó una regla y qué anotaciones se crearon.

II.3.5. Procesamiento de lenguaje natural para el reconocimiento e interpretación de expresiones temporales

Actualmente, existen dos aproximaciones diferentes a la hora de tratar la problemática de anotar y resolver las entidades temporales en textos escritos: basadas en Conocimiento y basadas en Aprendizaje computacional. A continuación se presenta cada una de ellas.

II.3.5.1. Sistemas basados en Conocimiento

Un sistema basado en Conocimiento puede definirse como un sistema que resuelve problemas para un dominio concreto utilizando una representación simbólica del conocimiento humano.

Las características principales de este tipo de sistemas son:

- Representación explícita del conocimiento
- Capacidad de razonamiento independiente de la aplicación específica
- Alto rendimiento en un dominio específico

Estas características se deben a la separación entre:

- Conocimiento específico del problema: Se trata de un conjunto de conocimientos aplicables al dominio concreto con que se está trabajando. Se pueden utilizar diferentes formalismos a la hora de representar dicho conocimiento aunque los sistemas, en el caso de la información temporal, definen el conocimiento como reglas. Es un elemento estático dentro de la arquitectura del sistema.
- Metodología para solucionar el problema: Proceso que efectúa el razonamiento a partir de los datos del problema que se intenta solucionar utilizando el conocimiento que posee. Esta metodología es genérica, es decir, qué se puede aplicar a diferentes dominios sólo cambiando el conocimiento que se utiliza.

El rendimiento de los sistemas basados en Conocimiento radica en la cantidad y la calidad del conocimiento de un dominio específico y no tanto de las técnicas de solución de problemas. Por tanto, si los sistemas de tratamiento de información temporal utilizan el enfoque basado en Conocimiento, necesitarán establecer todas las reglas necesarias para poder resolver las expresiones temporales de los textos y almacenar estas reglas en una base de conocimiento.

II.3.5.2. Sistemas basados en Corpus

Los sistemas basados en Corpus son un tipo de sistemas que desarrollan técnicas que aprenden a tratar o resolver una determinada tarea automáticamente. Estos sistemas suelen estar muy relacionados con métodos estadísticos ya que al igual que ellos se basan en el estudio de los datos.

Los algoritmos más comunes utilizados en este tipo de sistemas, que son aplicados también para el Procesamiento de Lenguaje Natural son:

- Aprendizaje supervisado: El sistema aprende de uno o varios corpus de ejemplo anotados previamente con la etiqueta correcta.
- Aprendizaje no supervisado: Se considera no supervisado cuando no existe tal anotación en los corpus y se utilizan otro tipo de recursos para aprender, como por ejemplo, diccionarios electrónicos, tesauros, etc. En concreto, el Aprendizaje Automático que se usa en la mayoría de los sistemas de tratamiento de información temporal será un aprendizaje supervisado que utiliza corpus anotados manualmente y supervisados para entrenar al sistema y así aprender automáticamente como resolver las diferentes expresiones temporales que aparecen en los textos.

Una vez decidida la aproximación a utilizar, las tareas que pueden realizarse para tratar los diferentes aspectos de la información temporal son:

- Anotación de las expresiones temporales.
- Ubicación en la línea temporal de los eventos de un texto asociados a una fecha concreta.
- Establecimiento de las relaciones temporales que existen entre los eventos que no tienen una fecha concreta asociada, indicando la relación de orden entre los mismos.

II.3.5.3. Esquemas de Anotación Temporal

A continuación, se detallan dos de los esquemas de anotación más destacados y utilizados (TIDES, TIMEML), haciendo especial hincapié en los elementos que cada uno de los esquemas es capaz de anotar así como los atributos asociados a cada uno de estos elementos.

TIDES

En los últimos tiempos se han creado multitud de corpus anotados para entrenar sistemas NLP. Durante este proceso de anotación surgió la necesidad de definir esquemas para hacer que los corpus resultantes fuesen explotables para tal fin. Uno de estos esquemos fue desarrollado por DARPA en una necesidad de anotar expresiones temporales que aparecen en los textos anotados. Este esquema se denomino TIDES (Ferro, et al. 2002) y ha sido aplicado ampliamente en sistemas de Búsqueda de Respuestas, Caracterización de Eventos, Resúmenes, etc. Este esquema de anotación está basado en la ontología temporal definida

por KSL (1991) aunque no es capaz de expresar todas las distinciones de la misma (pero si que se puede afirmar que lo definido en el esquema de anotación forma parte de la ontología).

El proceso de anotación de TIDES se divide en los pasos:

1. Marcar las expresiones temporales en un documento.
2. Identificar el valor temporal que la expresión está representando.

Las expresiones temporales se marcan utilizando un conjunto de disparadores léxicos con sentido temporal de tal forma que cualquier expresión que tenga alguno de estos disparadores es una expresión temporal.

Es importante remarcar que el esquema de anotación TIDES únicamente es válido para marcar expresiones temporales y las etiquetas que se utiliza para ello se denominan TIMEX2. Estas etiquetas tienen los siguientes atributos:

- VAL: se usa para representar el valor de una expresión que puede ser una instantánea en el tiempo o una duración.
- PERIODICITY: empleado para expresiones temporales que representan recurrencia regular (por ejemplo, el uso de términos como "siempre", "cada",...)
- MOD: usado como complementar de otros atributos en los casos en los que la expresión contiene algún modificador que puede cambiar su valor final, por ejemplo, la palabra "aproximadamente" antes de una expresión temporal.
- SET: indica si la expresión representa un conjunto de valores porque la expresión indica una repetición en el tiempo. A modo de ejemplo, la expresión "todos los lunes" tendría como valor del atributo SET un Sí.
- GRANULARITY: unidad temporal en la que está representado cada miembro del conjunto.
- NONSPECIFIC: booleano empleado para expresiones difusas y que por tanto no tienen una representación del atributo VAL concreta.
- COMMENT: facilita la inclusión de comentarios en cada etiqueta.

II.3.5.3.1. TIMEML

TimeML nacido en 2002 en la Institución ARDA y consiste en un esquema de anotación para anotar expresiones temporales, eventos y las relaciones temporales entre estos eventos (Radev & Sundheim, 2002).

Tiene bastantes similitudes con TIDES pero mejora éste manteniendo aquellas características que hacían de TIDES un esquema muy potente y añadiendo nuevas características, como la posibilidad de que un evento esté relacionado con más de un objeto indexado, o que un evento implique varias acciones.

Dentro del esquema TimeML existen cuatro grupos principales de etiquetas que a continuación se explican:

- EVENT: aquellas situaciones que ocurren se denominan eventos.
- TIMEX3: denotan tiempo como por ejemplo: "25 de octubre 1978" o "el próximo mes".
- SIGNAL: para anotar palabras qye denotan relación temporal entre objetos (por ejemplo, “durante”, “en”, etc.).
- LINK: para marcar la relación entre dos objetos, ya sea una relación temporal (TLINK), de subordinación (SLINK) o aspectual (ALINK).

Cada una de estas etiquetas puede tener a su vez diferentes atributos, que se muestran a continuación ordenados en función de tipo de la etiqueta:

Anotación EVENT

- ID: atributo obligatorio que contiene un identificador para cada evento.
- CLASS: los eventos se clasificación en función de la clase, según la siguiente organización:
    - PERCEPTION: implican la percepción física de otro evento (asociado comúnmnete a verbos como "mirar" o "escuchar").
    - REPORTING: eventos descritos por una persona u organización (muy común con el uso de verbos como "contar" o "explicar").
    - ASPECTUAL: indican el inicio de algo, el fin de algo, la culminación de algo o la continuación de algo (se emplean verbos como "empezar", "proseguir",etc.)
    - LACTION: empleado cuando un evento introduce algún argumento para describir una acción o situación de la que se puede inferir algo dada la relación que esta acción tiene con el evento LACTION. Un ejemplo sería "Algunas compañías están intentado monopolizar el negocio", donde "están intentando" sería un evento de tipo I_ACTION y "monopolizar" sería el argumento de evento que introduce.

  - LSTATE: Similar al anterior pero referido a estados en lugar de eventos (en este caso se emplearían expresiones como "esperar que", "opinar que", etc.)
  - STATE: describe estados.
  - OCCURRENCE: describe el resto de eventos que pueden ocurrir y que no se han contemplado en ninguno de los grupos anteriores.
- Atributos adicionales: también se pueden incluir información en la anotación mediante este atributo como el tiempo y el aspecto del verbo contenido en el evento.

Anotación TIMEX3

Está basada en la anotación de TIDES para expresiones temporales pero con algunas diferencias, como son el que no se usen los atributos SET, PERIODICITY, GRANULARITY y NON-SPECIFIC, aunque los tres primeros atributos se recogen enen una etiqueta denominada MAKEINSTANCE. Los atributos de TIMEX3 son:

- ID: atributo obligatorio para identificar la expresión temporal.
- TYPE: atributo obligatorio que indica el tipo de expresión al que se refiere la etiqueta (fecha, hora o duración).
- VAL: se usa para representar el valor de una expresión que puede ser una instantánea en el tiempo o una duración.
- MOD: usado como complementar de otros atributos en los casos en los que la expresión contiene algún modificador que puede cambiar su valor final, por ejemplo, la palabra "aproximadamente" antes de una expresión temporal.
- temporalFunction: atributo booleano que indica que la expresión temporal debe calcularse a través de una función.
- anchorTimelD: atributo opcional que indica el ID de otra expresión temporal con la cual está relacionada temporalmente.
- valuePromFunction: relacionado con el campo temporalFunction, representa el valor devuelto por la función empleada para calcular la expresión temporal.
- functionlnDocument: representa la función TIMEX3 que devuelve la fecha con la que la expresión temporal está relacionada y necesario para su resolución (la fecha de creación del documento, la fecha de modificación del documento,etc.)

Anotación SIGNAL

Denota la relación entre dos entidades temporales, ya sean éstas tiempos o eventos. Esta etiqueta sólo contiene un atributo ID.

Algunas de las señales que se pueden encontrar son:

- Preposiciones temporales como "en", "antes", "después", ...
- Conjunciones temporales como "mientras", "cuando", ...
- Modificadores temporales como "cada", "todos",...
- Expresiones negativas como "nunca",...
- Modales como "podría", "debería",...

Anotación LINK

- TLINK: marca una relación entre en evento y un tiempo o entre dos eventos. Está relación indica el orden de suceso entre los difentes eventos.
- SLINK: empleada cuando se encuentran verbos modales, negativas,… en el texto, este atributo indica el enlace de subordinación entre relaciones de eventos o eventos y señales.
- ALINK: indica la relación entre un evento de aspecto y algún argumento de dicho evento.

II.4. RECONOCIMIENTO DE VOZ

El reconocimiento de voz se define como la traducción del lenguaje hablado a texto, usando principalmente dos acrónimos para referirse al mismo: ASR (*Automatic Speech Recognition*) o STT (*Speech To Text*).

El reconocimiento de voz es un fenómeno muy complejo. Ni siquiera las personas comprendemos muy bien como se produce y como se percibe, teniendo una percepción muy simple de que el habla se construye a partir de palabras y éstas de fonemas. Sin embargo, la realidad es que el habla es un proceso dinámico sin partes que se puedan distinguir claramente (Huang & Acero & Hon, 2001).

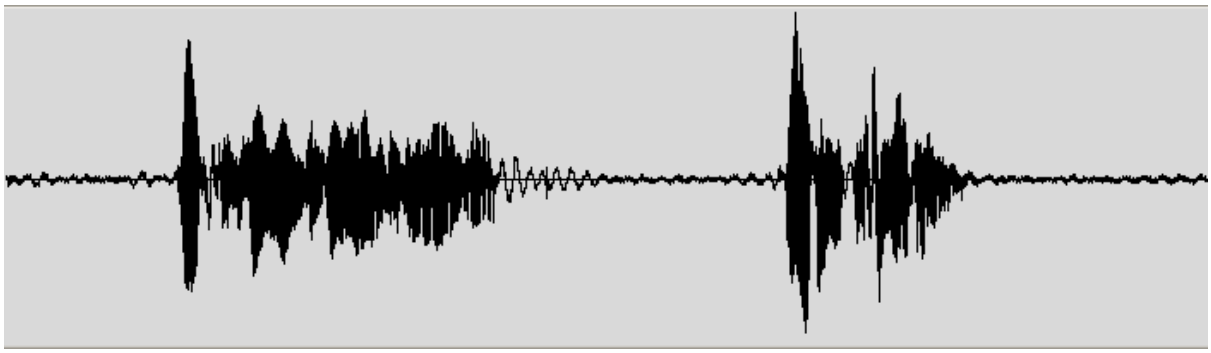


Figura 8. Ejemplo de grabación de voz

Actualmente, todas las aproximaciones modernas para crear sistemas de reconocimiento de voz son de alguna manera probabilísticas. Se entiende que no hay límites claros entre las unidades, o entre las palabras, lo que viene a significar que no existe una forma clara de delimitar cuando acaba una palabra y cuando empieza la siguiente, y es nuestro cerebro el que se encargar de hacer esa "partición" para ser capaces de comprender el significado de la frase. Esto mismo es lo que intentan hacer los sistemas de reconocimiento autómático del habla, y por desgracia no son capaces de conseguirlo con un 100% de precisión.

II.4.1. SU USO EN LA INDUSTRIA

El reconocimiento de voz se emplea mayoritariamente en los sistemas de telefonía convencional, donde se emplea esta tecnología para derivar a la persona que llama a la información o persona que desea. Es común el uso de esta tecnología en sistemas IVR (*Instant Voice Response*) donde la persona llama a una centralita automática que contesta empleando una voz artificial o leyendo un mensaje pregrabado, e invita a la persona a decir qué desea ofreciendo en algunos casos una lista de opciones posibles o bien la posibilidad de responder con libertad.

Este uso ahorra una gran cantidad de costes a las empresas que deben dar soporte técnico, comercial, o algún tipo de servicio telefónico a un gran número de usuarios, porque permite resolver muchas cuestiones sin necesidad de un operador.

Otro contexto donde se ha empleado mucho el reconocimiento de voz es en la medicina (Suominen & Zhou & Hanien & Ferraro, 2015), donde se emplean sistemas de dictado automático para transcribir los informes médicos. Esto es una práctica muy común en especialidades médicas donde se los médicos han empleado ampliamente el uso de grabadoras de voz y de transcriptores humanos. Ahora, ese proceso se ha automatizado

mediante sistemas de reconocimiento de voz especialidades que obtienen la transcripción con un elevado grado de precisión e incluso en tiempo real, lo que permite al médico tener el informe realizado inmediatamente terminar de dictar.

También se debe destacar el uso del reconocimiento de voz en entornos militares, donde hay muchos casos de sistemas empleados en aviones y helicópteros de combate.

Sin embargo, en la actualidad el reconocimiento de voz es algo extendido gracias a la incorporación de esta tecnología en los *smartphones*. Actualmente los grandes fabricantes de sistemas operativos para telefónicos móviles incorporan tecnología de reconocimiento de voz para realizar diversas acciones como dictar mensajes o realizar preguntas sobre un determinado tema. Ejemplos de estos sistemas son Apple Siri, Google Now o Microsoft Cortana.

II.4.2. ESTRUCTURA DEL HABLA

En el estado del arte actual, el habla se trata como un *stream* continuo de audio dentro de una máquina de estados donde coexisten estados estables y estados dinámicos (Rabiner, 1993). En esa secuencia de estados, cada uno puede representar un conjunto de sonidos más o menos similares, o fonemas. Las propiedades acústicas de la onda de voz asociadas a un fonema pueden variar enormemente dependiendo de diferentes factores: contexto, hablante, estilo del habla, etc. Estas variaciones son claras puesto que se aprecian al escuchar gente de diferentes países o regiones hablando el mismo idioma. A este fenómeno se le denomina “coarticulación”, y viene a representar las diferencias de pronunciación de los fonemas con su forma canónica (Rabiner, 1993).

En la máquina de estados, las transiciones entre palabras ofrecen más información que las transiciones entre regiones o fonemas estables, por lo que en el desarrollo de sistemas de reconocimiento de voz se suelen emplear “difonemas” (fonemas parciales entre dos fonemas consecutivos), aunque a veces se habla de unidades “subfonéticas” que representan diferentes subestados de un fonema. Este razonamiento hace que se hable normalmente de 3 estados cuando se procesa un fonema: la primera parte del fonema depende de su fonema predecesor, la parte intermedia es la estable y la siguiente parte dependen del siguiente fonema.

En muchas ocasiones los fonemas se evalúan dentro de un contexto. Estos fonemas se denominan trifonemas o quinfonemas y permiten formar palabras completas a partir de sus fonemas.

Desde el punto de vista computacional (Jelinek, 2001), es útil detectar partes de trifonemas en vez de los trifonemas completos. Toda la variedad de detectores de sonidos puede ser representada de esta forma por un pequeño conjunto de detectores de diferentes sonidos breves. Normalmente, se usan alrededor de 4000 detectores de sonidos cortos para componer detectores de trifonemas. A esos detectores se los denomina "senones", y suelen definirse mediante árboles de decisión u otras estructuras similares debido a su complejidad.

Según lo explicado anteriormente, se puede afirmar que los fonemas construyen unidades de "subpalabras", como las sílabas. En ocasiones, las sílabas se definen como "entidades de reducción estable". Por ejemplo, cuando se habla rápido, los fonemas cambian frecuentemente, pero las sílabas no. Además, las sílabas están relacionadas con la entonación.

Tras esto, llegamos a las palabras. Éstas son importantes en el reconocimiento de voz porque restringen las combinaciones de fonemas significativamente. Por ejemplo, si un idioma tiene 40 fonemas y la media de una palabra tiene 7 fonemas, puede haber entonces 40^7 palabras posibles. Sin embargo, lo normal es que una persona que tenga un vocabulario rico no empleé más de 20.000 palabras, lo que hace que el reconocimiento de voz sea más sencillo.

Las palabras y otros sonidos sin significado lingüístico (respiración, titubeos, balbuceos,…) forman lo que en las investigaciones acerca del reconocimiento de voz llaman "utterances". "utterances" se pueden definir como fragmentos separados de audio entre pausas, por lo que no tienen que representar necesariamente una frase, que es un concepto más semántico.

II.4.3. PROCESAMIENTO DEL HABLA

El reconocimiento del habla se realiza procesando la onda de voz, identificando las partes del audio con habla entre silencios (*utterances*) e intentando reconocer el habla en esos fragmentos. Para esta última parte es necesario tener en cuenta todas las posibles combinaciones de palabras e intentar corresponder las mismas con el audio, para lo que se

usan técnicas como el *best matching combination*, como se comenta a continuación (Yu & Deng, 2014).

En primer lugar, en el reconocimiento del habla se extraen diferentes características que son las que ayudan a realizar el proceso. El habla es divida en *frames* normalmente de 10 milisegundos para extraer 39 valores que representan el habla, y que forman el vector de características. La forma de obtener esas características es motivo de investigación actual pero para simplificar se puede decir que proviene del espectro de la onda.

Por otro lado, existe un modelo matemático que recoge los atributos comunes de una palabra hablada, y que en la práctica suele hacerse empleando el modelo gausiano mixto de tres estados más probable para representar un senone.

En la actualidad, la mayoría de sistemas de reconcimiento de voz usan Modelos Ocultos de Markov (HMM), que representan un modelo genérico descrito como una secuenca de estado que cambian de uno a otro en función de una probabilidad calculada con anterioridad (Jelinek, 2001). Este modelo es capaz de describir cualquier proceso secuencial como es el caso del habla, y se ha demostrado que funciona muy bien en reconocimiento de voz. Sin embargo, empiezan a aparecer sistemas de reconocimiento de voz que están usando técnicas de Redes Neuronales Profundaas (DNN) y que están consiguiendo resultados muy prometedores. Sobre estos puntos se habla más adelante en este documento.

II.4.4. MODELOS EN EL RECONOCIMIENTO DE VOZ

En el reconocimiento de voz es necesario emplear tres diferentes modelos para ser capaces de reconocer el habla (Pieraccini, 2012):

1. Modelo acústico: contiene la información sobre las propiedades acústicas de cada senone. Dentro de los modelos acústicos están los que son independientes del contexto, que ofrecen los vectores de características más probables para cada fonema, y los dependientes del contexto, que están construidos a partir de los senones obtenidos en un contexto.
2. Diccionario fonético: se trata de un mapeo de palabras a fonemas. Este mapeo no es muy efectivo debido a las variaciones del habla que existen entre diferentes hablantes debido a motivos de sexo, edad, localización, etc. Normalmente se tienen hasta 3 variaciones de la pronunciación de una misma palabra, pero pueden existir muchas

más, aunque con esas variaciones suele bastar en la mayoría de los casos para ser capaces de corresponder una pronunciación determinada con su correspondiente palabra.

3. Modelo lingüístico: se usa para restringir la búsqueda de palabras. Define qué palabras pueden seguir a las palabras reconocidas anteriormente, lo que hace que se acote signficativamente el rango de palabras posibles. Se basa en el razonamiento de que unas palabras tienen más probabilidad de aparecer en un contexto que otras, e incluso que hay palabras que no tienen sentido en un contexto y por tanto es muy improbable que el hablante las haya dicho. La mayoría de modelo lingüísticos emplean n-gramas qe contienen información estadísticas de secuencias de palabras en un modelo finito de estados. Estos modelos se crean analizando grandes cantidades de texto del dominio de forma que se indica la probabilidad que tiene una palabra en aparecer después del resto de n-1 palabras existentes. En la siguiente figura se muestra un pequeño fragmento de un fichero ARPA-LM del modelo lingüístico en portugués del producto INVOX Medical Dictation (http://www.invoxmedical.com) a modo ejemplo de este modelo n-gram:

```
-0.470928   LABORATORIALMENTE   E   A
-1.711383   LABORATORIALMENTE   PONTO   JP
-0.279841   LABORATORIALMENTE   PONTO   NÃO
-0.425969   LABORATORIALMENTE   PONTO   RIM
-0.410353   LABORATORIALMENTE   VÍRGULA COM
-0.410353   LABRAIS ÂNTERO  SUPERIOR
-0.470928   LABRAIS VÍRGULA SEM
-0.410353   LABRAL  ANTERIOR    VÍRGULA
-0.470928   LABRAL  CONCLUSÃO   DOIS
-0.410353   LABRAL  E   CISTO
-0.410353   LABRAL  LIGAMENTO   REDONDO
-0.771958   LABRAL  PONTO   A
-1.012413   LABRAL  PONTO   BURSA
-1.012413   LABRAL  PONTO   TENDÃO
-0.470928   LABRAL  RX  DE
```

Figura 9. Fragmento de modelo lingüístico de en formato ARPA-LM para el portugués.

Estos tres modelos se combinan en un motor de reconocimiento de voz. Estos motores son realmente independientes del idioma en el que se van a emplear, y son estos tres modelos lo que hacen que el sistema sea adecuado para un contexto y/o idioma determinado.

II.4.5. ALGORITMOS EMPLEADOS EN EL RECONOCIMIENTO DE VOZ

Los modelos acústicos y lingüísticos son partes fundamentales de cualquier sistema de reconocimiento de voz basado en algoritmos estadísticos. Los Modelos Ocultos de Markov (HMM) son ampliamente usados en muchos sistemas, pero existen otras aproximaciones, algunas de las cuales están cogiendo fuerza en los nuevos sistemas que han aprecido.

II.4.5.1. Modelos Ocultos de Markov (HMM)

Como se ha mencionado anteriormente, la mayoría de los sistemas modernos de reconocimiento de voz se basan en HMM (Jelink, 2001). Estos son modelos esta´disiticos que proporcionan como salida una secuencia de símbolos o cantidades. Los HMM se usan en el reconocimiento de voz porque una señal de voz puede verse como señal estacionaria. En una escala corta de tiempo, como la de 10 milisegundos usada ampliamente como *frame* en el procesamiento de la señal del habla, la voz puede aproximarse como un proceso estacionario. Según esto, la voz puede interpretarse como un modelo de Markov para muchos propósitos estocásticos.

Otra razón por lo que los HMM son tan populares en el reconocimiento de voz (Schroeder, 2004) se debe a que pueden ser entrenados automáticamente de manera sencilla y abordable computacionalmente hablando. En el reconocimiento de voz, un HMM puede devolver un secuencia de valores reales *n*-dimensional de vectores (normalmente $n = 10$), devolviendo dichos vectores cada 10 milisegundos, o en función del tamaño de *frame* seleccionado. Los vectores consisten en coeficiones cepstrales que se obtienen mediante transformadas de Fourier del *frame* de audio analizado y correlacionando el espectro de la señal mediante una transformada de coseno, obteniendo los coeficientes más significativos. Los Modelos Ocultos de Markov tienden a tener una distribución estadística que es una mezcla de la covarianza diaginal gausiana en cada estado. Cada fonema tendrá entonces una distribución diferente, y el HMM para una secuencia de fonemas se realiza concatenando los HMM individuales.

En la actualidad, los sistemas de reconocimiento de voz modernos emplean la combinación de diversas técnicas para mejorar los resultados del reconocimiento (Piernaccini, 2012). Un sistema de reconocimiento de voz con un vocabulario muy extenso necesitará definir fonemas dependientes del contexto, para lo que se usará normalización cepstral para normalizar las diferencias entre hablantes y condiciones acústicas.

II.4.5.2. *Dynamic time warping* (DTW)

Esta es la aproximación clásica empleada por los sistemas de reconocimiento de voz más antiguos, pero que ha sido reemplazada en la mayoría de casos por los métodos basados en HMM. Sin embargo, se ha considerado interesante mencionarlo.

DTW es un algoritmo para mediar la similaridad entre dos sencuencias que pueden variar en tiempo o velocidad (Rabiner, 1993). Por ejemplo, puede usarse para ver similaridades entre distintas formas de andar, de teclear un código, etc.

Durante mucho tiempo, ha sido la técnica empleada por los sistemas de reconocimiento de voz para cubrir las diferentes formas de hablar. En general, es un método que permite encontrar una correspondencia óptima entre dos sencuencias pero tiene importantes restricciones, que han hecho que los desarrolladores de sistemas de reconocimiento de voz se decanten por los HMM.

II.4.5.3. Redes neuronales

Las redes neuronales empezaron a usarse en la construcción de modelos acústicos a finales de los años 80 (Beigi, 2007). Desde entonces, las redes neuronales se han empleado en muchos campos del reconocimiento de voz como la clasificación de fonemas, el reconocimiento aislado de palabras y en la adaptación del hablante, lo que se conoce comúnmente como “entrenamiento de voz” y que permite mejorar la precisión del sistema para un hablante específico tras hacer que éste lea un texto predeterminado.

A diferencia de los HMM, las redes neuronales no hacen suposiciones sobre las propiedades estadísticas o características, y son capaces de realizar entrenamientos discriminativos de una manera muy eficiente, haciendo esta técnica muy atractiva. Sin embargo, a pesar de ser muy efectivas para clasificar unidades cortas de audio, como fonemas individuales o palabras aisladas, las redes neuronales no han conseguido ser efectivas para reconocimiento continuo del habla, principalmente por la imposibilidad de modelar dependencias temporales.

En cualquier caso, recientes investigaciones empleando redes neuronales más específicas como las Redes Neuronales Recurrentes (RNN) y las Redes Neuronales con Retardo de Tiempo (TDNN) han vuelto a poner de moda el uso de estas técnicas en el reconocimiento de voz (Yu & Deng, 2014), porque han conseguido ser capaces de identificar dependencias temporales y ysar esa información para mejorar tareas de reconocimiento de voz. Sin embargo, este tipo

de redes incrementan mucho el coste computacional de estos sistemas y hacen que el reconocimiento de voz sea muy lento, por lo que se está investigando activamente en la actualidad para salvar esos problemas.

Igualmente, se están empezando a usar Redes Neuronales Profundas (DNN) y *Denoising Autoencoders* para abordar los problemas de las redes neuronales anteriormente mecionadas, y en la actualidad el uso de DNN para los sistemas de reconocimiento de voz está siendo muy prometedor, siendo la elección de Microsoft o Google para desarrollar sus sistemas actuales de reconocimiento de voz en la nube.

Debido a todas estas limitaciones, hay muchos sistemas de reconocimiento de voz que siguen empleando HMM como técnica principal, pero que usan redes neuronales para otro tipo de tareas de preprocesamiento como transformación de características, reducción de dimensionalidad, etc.

II.4.5.4. Redes Neuronales Profundas (DNN)

Una red neuronal profunda (DNN) es una red neuronal con multiples capas ocultas entre la capa de entrada y de salida de la red (Yu & Deng, 2014).

Las DNN pueden modelar complejas relaciones no lineales, y sus arquitecturas generan modelos de composición donde las capas "extra" permiten la composición de características de las capas anteriores, ofreciendo una inmensa capacidad de aprendizaje y por tanto un importante potencial en el modelado de complejos patrones del habla. Las DNN son el tipo más popular de arquitecturas de aprendizaje profundas que se emplean en la construcción de modelos acústicos para el reconocimiento de voz desde el 2010 (Pieraccini, 2012).

Las DNN han sido probadas con éxito en sistemas de reconocimiento de voz con grandes vocabularios en estados HMM dependientes del contexto construidos mediante árboles de decisión.

Las DNN se están usando en reconocimiento de voz obteniendo grandes progresos en las siguientes áreas:

1. Escalado y aceleración de los procesos de entreamiento y decodificación de la DNN.
2. Entrenamiento secuencial discriminativo de DNN.
3. Procesamiento de características por modelos profundos con comprensión sólida de los mecanismos subyacentes.
4. Adaptación de las DNN y de los modelos profundos relacionados.

5. Multitarea y transferencia de aprendizaje por DNN y otros modelos profundos relacionados.
6. Diseño y evolución de redes neuronoales para explotar dominios concretos de reconocimiento de voz.
7. Redes neuronales recurrentes y sus variantes LSTM.
8. Otro tipo de modelos profundos incluyendo modelos discriminativos.

En los últimos años, las DNN están teniendo mucho éxito en los congresos especializados en reconocimiento de voz, como IEEE-ICASSP e Interspeech, donde se puede ver como el número de artículos aceptados relacionados con este tema ha crecido exponencialmente desde 2010. Además, muchos sistemas comerciales de reconocimiento de voz como Microsoft Cortana, el empleado en la XBOX, Skype Trnaslator, Google Now, Apple Siri,… están basados en la actualidad en este tipo de técnicas.

II.4.6. OPTIMIZACIÓN DE LOS SISTEMAS DE RECONOCIMIENTO DE VOZ

Los sistemas de reconocimiento de voz suelen dar resultados pobres cuando se intentan utilizar de forma genérica, aunque esto está cambiando gradualmente y ya se cuenta con sistemas que ofrecen resultados aceptables en entornos multicontexto. Esto se puede apreciar en la tecnología de reconocimiento de voz que ofrecen los *smartphones* actuales, que permiten dictar mensajes con una precisión bastante elevada. Sin embargo, estos mismos sistemas ofrecen una tasa de acierto muy pobre cuando se emplean para cosas más específicas, como es el caso del dictado médico.

Por estas razones, es importante que se realicen mejoras y optimizaciones para el contexto donde se va a aplicar el reconocimiento de voz. Estas mejoras consisten en optimizar los modelos de los que se ha hablado anteriormente, de forma que el sistema de reconocimiento de voz esté preparado para entender bien el habla conociendo el vocabulario que el usuario empleará y las características acústicas del entorno (acentos, ruido ambiente, etc).

Para mejorar un sistema de reconocimiento de voz normalmente se emplean las siguientes medidas que permiten evaluar de una forma cuantitativa y objetiva la calidad del reconocimiento de voz (Deng & Li & Kwan & Stern, 2007):

- Ratio de error por palabras (WER): se compara el texto original con el texto reconocido. Suponiendo que hay N palabras, se tienen en cuenta aquellas palabras

añadidas erróneamente (*I*), las que no se han transcrito (*D*) y las palabras que no se transcribieron correctamente (*S*) de forma que el WER se define como:

$$\text{WER} = \frac{I+D+S}{N}$$

Figura 10. Fórmula de cáculo del *Word Error Rate* (WER).

- Precisión: muy parecido a la anterior pero no tiene en cuenta las inserciones:

$$\text{Precisión} = \frac{N-D-S}{N}$$

Figura 11. Fórmula de cálculo de la precisión.

- Velocidad: representa la capacidad del sistema de procesar el audio para obtener la transcripción. La unidad de medida se representa como "RT" (*Real-Time*) indicando la relación entre la duración del audio procesado y el tiempo del proceso de transcripción. Por ejemplo, si en procesar 10 minutos de audio se necesitan 5 minutos, el sistema tiene una velocidad de 0,5RT. En la actualidad, la mayoría de sistemas de reconocimiento de voz existentes en el mercado permiten operar por debajo de 1RT.

II.5. CLOUD COMPUTING

El trabajo que plantea esta tesis contempla una solución software basada en servicios en la nube para crear interfaces de diseño para todos empleando tecnologías de reconocimiento de voz y procesamiento de lenguaje natural.

Puesto que el despliegue de los sistemas desarrollados en este trabajo se han diseñado para ser empleados en este tipo de infraestructura, se ha considerado interesante añadir un apartado de este capítulo dedicado a enumeran las características del *Cloud Computing* y un estudio de los distintos proveedores consultados durante el desarrollo de este trabajo.

II.5.1. CARACTERÍSTICAS DEL CLOUDCOMPUTING

La computación en la nube se ha convertido en uno de los factores relevantes para las empresas, desarrolladores y trabajadores, porque proporciona herramientas y aplicaciones Web que permite almacenar información en servidores externos. Además, la computación en la nube ofrece ventajas tales como: reducción de costos, acceso a la información desde cualquier lugar, por mencionar sólo algunas [1].

El Instituto Nacional de Estándares y Tecnologías (NIST) de Estados Unidos define computación en la nube como un modelo para permitir un acceso de red adecuado, desde cualquier sitio y bajo demanda a un conjunto compartido de recursos informáticos configurables (por ejemplo, redes, servidores, almacenamiento, aplicaciones y servicios) que se pueden proporcionar rápidamente y lanzar con un mínimo esfuerzo de gestión o interacción con el proveedor de servicios [2].

En la actualidad existen tres tipos de nubes: públicas, privadas e hibridas. Una nube pública o externa, se caracteriza por tener las aplicaciones y servicios Web de un proveedor disponible a los clientes a través de internet. Las nubes privadas o internas se centran en el uso exclusivo de una sola organización que tiene muchos consumidores o usuarios. Además, la nube hibrida combina los modelos de nubes públicas y privadas. Por otro lado, existen tres modelos de servicio de la nube: 1) SaaS (Software as Service), 2) PaaS (Platform as a Service) and 3) IaaS (Infrastructure as a Service). IaaS ofrece una infraestructura de recursos (almacenamiento, base de datos, entre otros), generalmente en términos de máquina virtual como un servicio. PaaS ofrece un entorno orientado al desarrollo, prueba, despliegue y alojamiento de aplicaciones. SaaS ofrece un conjunto de aplicaciones instaladas y ejecutándose en internet. Hoy en día, hay varios proveedores de computación en la nube como: Google Apps, Zoho, AppEngine, E2C Amazon, entre otros. Estos proveedores ofrecen software, infraestructura o plataforma como un servicio.

En este apartado se realiza una comparación de las características que ofrece cada una de los proveedores de servicio PaaS de java, clasificados en cuatro áreas: 1) Soporte de tecnologías para desarrollo basado en Java, 2) Soporte al proceso de desarrollo y a la productividad del desarrollador, 3) Características arquitectónicas, y 4) Decisiones de negocio.

II.5.2. ESTUDIO DE PROVEEDORES DE CLOUDCOMPUTING

En este apartado se muestra un listado con los distintos proveedores evaluados y las características de cada uno de ellos.

II.5.2.1. Amazon Elastic Beanstalk

Elastic Beanstalk permite gestionar e implementar aplicaciones en la nube de Amazon Web Services (AWS). Elastic Beanstalk utiliza servicios de AWS tales como Amazon Elastic Cloud

Compute (Amazon EC2), Amazon Simple Storage Service (Amazon S3), Amazon Simple Notification Service (Amazon SNS), Elastic Load Balancing y Auto Scaling, ofreciendo así una infraestructura fiable, escalable y rentable. Además, Elastic Beanstalk gestiona de manera automática los detalles de implementación, aprovisionamiento de capacidad, equilibrio de carga, auto escalado y gestión del estado de la aplicación. Elastic Beanstalk proporciona un control absoluto sobre los recursos de AWS que potencian las aplicaciones. Por último, cabe mencionar que Elastic Beanstalk garantiza una portabilidad sencilla de la aplicación debido a que se compila mediante pilas de software conocido, como HTTP Server para Node.js, PHP y Phyton, Passenger para Ruby, IIS 7.5 para .NET y Apache Tomcat para Java.

II.5.2.2. AppFog

AppFog es una plataforma en la nube para aplicaciones web construida sobre Cloud Foundry. AppFog ofrece una arquitectura escalable con funciones de almacenamiento en caché y equilibrio de carga. AppFog provee una interface gráfica de trabajo, una línea de comando y una API REST para el desplegar, monitorear y escalar aplicaciones basadas tecnologías tales como Java, Node, Ruby, Python, PHP y .NET, así como bases de datos MySQL, PostgreSQL, MongoDB, Redis, y servicios RabbitMQ. A través de la consola de AppFog, es posible crear equipos para colaborar en un proyecto, buscar software de terceros (Add-on) para extender los servicios de la aplicación, así como administrar el uso de memoria de las aplicaciones y las instancias. Además, AppFog permite el despliegue de aplicaciones en diversas nubes, tales como Amazon, Rackspace, Microsoft Windows Azure, entre otras.

II.5.2.3. Apprenda

Apprenda es una plataforma en la nube para aplicaciones basadas en .Net y Java. Apprenda ofrece capacidades de escalamiento automático o manual de los componentes de las aplicaciones a través de interfaces de usuario, línea de comandos, y la API. Apprenda utiliza una arquitectura distribuida que permite alojar aplicaciones Web y SOA de forma descentralizada, permitiendo así tolerancia a fallos, fácil escalabilidad y alta disponibilidad.

II.5.2.4. Cloud Foundry

Cloud Foundry es una plataforma de código abierto desarrollada por VMWare bajo la licencia Apache 2.0. Ésta plataforma permite desarrollar, desplegar y escalar aplicaciones en

la nube. Actualmente, Cloud Foundry provee soporte para lenguajes Java, Grrovy, Scala, Ruby y Node.js, así como para diversos marcos de trabajo tales como Spring, Rails, Grilas, por mencionar algunos. Además, Cloud Foundry soporta las infraestructuras vSphere, vCloud, OpenStack, y Amazon AWS a través de la herramienta BOSH.

II.5.2.5. CloudBees

CloudBees permite construir, desplegar y administrar aplicaciones en la nube basado en cualquier lenguaje basado en la máquina virtual de Java tales como JRuby, Grails, Scala, Groovy, Java entre otros. CloudBees permite integración continua a través de Jenkins logrando así un ágil desarrollo y despliegue de aplicaciones. Además, CloudBees provee una única plataforma híbrida lo que permite administrar servicios en la nube pública o en un centro de datos propietario.

II.5.2.6. CloudControl

CloudControl es una plataforma europea que permite desarrollar y desplegar aplicaciones en la nube. Dicha plataforma permite agregar funcionalidad o servicios a través de “Add-ons”, dichos servicios incluyen tareas tales como manejo de base de datos, almacenamiento en caché, supervisión del rendimiento, entre otros. Otro punto importante de esta plataforma es que permite tanto el monitoreo del desempeño, como las pruebas de carga con el objetivo de determinar las opciones óptimas para escalar horizontal o verticalmente la aplicación.

II.5.2.7. Cloudify

Cloudify es una plataforma que permite desarrollar, administrar y escalar existentes y nuevas aplicaciones sobre cualquier nube. El enfoque de Cloudify permite instalar aplicaciones a través de comandos REST. Con respecto a la migración de aplicaciones, Cloudify provee recursos necesarios para desplegar la aplicación en cualquier nube ya sea pública, privada, o incluso híbrida. Además, Cloudify provee herramientas para monitorear la disponibilidad y desempeño de las aplicaciones, así como definir métricas personalizadas a usar, garantizando así la continuidad de la aplicación en todo momento. Clodify aborda el tema de elasticidad permitiendo definir reglas de escalabilidad basadas en cualquier métrica personalizada, y a las cuales la plataforma responderá llevando a cabo el escalamiento vertical u horizontal necesario.

II.5.2.8. CumuLogic

CumuLogic es una plataforma escalable para el desarrollo y despliegue de aplicaciones móviles y web en cualquier nube. CumuLogic provee servicios integrados en la nube tales como Bases de datos-como-Servicio y Cache Distribuida-como-Servicio. Dichos servicios permiten a las empresas el desarrollo, despliegue y producción de aplicaciones web, servicios back-end de aplicaciones móviles, servicios web, la recuperación de desastres, la migración de carga de trabajo a la nube, la consolidación de aplicaciones, y el análisis de datos, entre otros. CumuLogc permite la migración de aplicaciones existentes a la nube con mínimo o ningún cambio en el código. Además, CumuLogic provee servicios en la nube compatibles como Amazon Web Services, permitiendo la fácil implementación de una estrategia de nube híbrida. Po último, cabe mencionar que CumuLogic provee soporte para todo el ciclo de vida de la aplicación, desde el desarrollo, hasta las pruebas y despliegue de la misma.

II.5.2.9. DotCloud

dotCloud es una plataforma como servicio para hospedaje de aplicaciones basadas en Java, PHP, JavaScript, Phyton, Ruby y Perl que aprovecha el motor de contenedores de Linux Docker para ofrecer soluciones de nube pública, privada e híbrida.

II.5.2.10. Engine Yard

Engine Yard es una plataforma como servicio que automatiza la configuración, despliegue y la gestión continua de las aplicaciones en la nube. Engine Yard se lanzó en 2006 como PaaS enfocada en Ruby on Rails, JRuby, PHP, Node.js y Java. Es compatible con varios proveedores de IaaS, por lo que los clientes pueden elegir la mejor infraestructura para sus aplicaciones. Engine Yard se ha asociado con Amazon Web Services, Verizon Terremark y Microsoft Windows Azure.

II.5.2.11. Google App Engine

Google App Engine es una plataforma como servicio que permite desarrollar y ejecutar aplicaciones sobre la infraestructura de Google. Las aplicaciones generadas son fáciles de escalar a medida que sus necesidades de tráfico y almacenamiento de datos cambian. Google App Engine soporta el desarrollo de aplicaciones en una variedad de lenguajes de programación tales como Java, Python, PHP y Go. Google App Engine incluye, entre otras

características: almacenamiento persistente con consultas, ordenamiento, y transacciones, balance de carga y escalamiento automático, colas de tareas asíncronas para la realización de tareas fuera del ámbito de una solicitud, tareas programadas para activar eventos en momentos o intervalos regulares especificados, y la integración con otros servicios y cloud APIs de Google. Todas las aplicaciones desarrolladas con Google App Engine se ejecutan en un ambiente sandbox seguro, lo que permite la distribución de solicitudes a través de múltiples servidores, y la ampliación de servidores para satisfacer las demandas de tráfico. Además, Google App Engine proporciona un SDK (Software Development Kit) que permite administrar la aplicación a nivel local, y una consola de administración que maneja la aplicación en la producción. Dicha consola utiliza una interfaz basad en la Web para entre otras cosas, crear nuevas aplicaciones, configurar los nombres de dominio, y examinar loa archivos de registro de acceso y errores.

II.5.2.12. Heroku

Heroku es una plataforma que permite el desarrollo, despliegue y administración de aplicaciones Web basadas en los lenguajes de programación Ruby, Node.js, Java, Python, Clojure y Scala. Heroku ofrece un conjunto de herramientas de desarrollo y servicios específicos de la aplicación en una única plataforma para desarrollar, probar, implementar y administrar el proceso de desarrollo de aplicaciones. Heroku se compone de varios componentes que trabajan juntos para proporcionar una plataforma completa de desarrollo de nubes. Por ejemplo APIs proporcionan gestión de versiones, colaboración y gestión de control de acceso, escalabilidad y otras actividades de gestión de operación.

II.5.2.13. Jelastic

Jelastic es una plataforma como infraestructura (*Platform as Infrastructure, PaI*) (una combinación de plataforma como servicio e infraestructura como servicio) para aplicaciones basadas en Java, PHP y Ruby. Jelastic provee soluciones para desarrolladores, empresas y proveedores de hospedaje Web como tres productos diferentes: Jelastic for Developers (nube pública), Jelastic for Enterprise (nube privada) y Jelastic for Hosting (nube hibrida), respectivamente. Jelastic depende de la infraestructura subyacente de alrededor de 20 proveedores de infraestructura como servicio alrededor del mundo; por lo tanto, los costos pueden variar de acuerdo al proveedor seleccionado. De hecho, las tres ediciones de Jelastic

están disponibles como productos comerciales, aunque también existen versiones de prueba para cada edición.

II.5.2.14. Red Hat OpenShift

Red Hat OpenShift is es una plataforma como servicio de código abierto (en parte) para desarrollar y hospedar aplicaciones basadas en Java, PHP, Ruby, JavaScript, Phyton y Perl; que ofrece tres productos distintos: OpenShift Online (nube pública), OpenShift Enterprise (nube privada) y OpenShift Origin (nube hibrida). Red Hat OpenShift utiliza un mecanismo de extensión basado en *cartriges* lo que le otorga facilidades para integrar servicios de terceros como *backends* externos, frameworks, Database Management Systems (DBMS), entre otros. Asimismo, dicho mecanismo proporciona cierto nivel de portabilidad entre proveedores PaaS a las aplicaciones en OpenShift. Las ediciones Online y Origin de OpenShift son gratuitas, mientras que la edición Enterprise es un producto comercial, aunque existe una versión de prueba disponible.

II.5.2.15. Salesforce

Salesforce1 platform es, formalmente, la combinación de dos componentes principales: las plataformas como servicio Force.com y Heroku1 (versión para Salesforce de Heroku). Force.com es una plataforma enfocada en *social enterprise* para desarrollo de aplicaciones Web, de escritorio y móviles, utilizando un lenguaje de programación de propósito no general llamado Apex y un lenguaje de marcado basado en XML para desarrollo de interfaces de usuario llamado VisualForce. Heroku1 es una plataforma para hospedaje de aplicaciones basadas en Java, Ruby, JavaScript and Phyton que permite conectar con datos en Salesforce. Heroku1 utiliza *buildpacks* para construir (compilar y desplegar) aplicaciones directamente en la nube; este mecanismo es considerado como un medio para dotar a las aplicaciones de cierta portabilidad como el caso de los *cartdriges* de Red Hat OpenShift.

II.5.2.16. Stackato

Stackato es una plataforma empresarial (nube privada) para desarrollo y hospedaje de aplicaciones basadas en Java, JavaScript, Perl, PHP, Phyton and Ruby (por defecto). Stackato permite dar soporte a prácticamente cualquier otro lenguaje mediante el uso de *buildpacks* (Heroku) dada su arquitectura basada en *ad-ons*; este mecanismo dota a las aplicaciones en

Stackato de cierto grado de portabilidad a nivel de datos. Stackato está construido sobre componentes open source como la plataforma como servicio CloudFoundry y el motor de contenedores para Linux Docker. Stackato ofrece además de la edición empresarial, una edición de nube pública denominada Stackato Micro Cloud, la cual está disponible como software gratuito. Esta versión, sin embargo, está limitada en servicios de soporte y en funcionalidad.

II.5.2.17. Windows Azure

Microsoft Windows Azure es una plataforma como servicio basada en .NET para desarrollo y hospedaje de aplicaciones y sitios Web, aplicaciones móviles y servicios Web como servicios de datos y de mensajería (middleware). Windows Azure soporta los lenguajes JavaScript, Java, PHP, Phyton, Ruby y Objective-C, además de .NET gracias al mecanismo de extensión conocido como *worker rol*. Windows Azure integra características de infraestructura como servicio tales como máquinas virtuales en Windows Server y Linux. Los costos de uso de la plataforma varían en función del tipo de servicio utilizado. En cualquier caso existe una versión de evaluación de un mes que proporciona un presupuesto de aproximadamente $200. Los servicios que conforman la plataforma Windows Azure permite implementar no solo aplicaciones de nube pública sino también aplicaciones hibridas (nube pública-privada).

II.5.2.18. WSO2 StratosLive

WSO2 StratosLive es una plataforma como servicio pública y de código abierto para desarrollo y hospedaje de aplicaciones Web y móviles basadas en Java. StratosLive está construida sobre la plataforma de *middleware* Stratos de WSO2 (nube privada) y comparte una base de código común con la plataforma de *middleware* empresarial Carbon también de WSO2 (nube hibrida). StratosLive también ofrece soporte para desarrollo basado en PHP, C, C++, Perl, Ruby y Python. StratosLive integra un servidor de aplicaciones Web propio y *cloud native* basado en Apache Axis2; sin embargo, es posible integrar a la plataforma servidores Web externos como Tomcat. StratosLive está disponible en cuatro ediciones distintas: demo, SMB, profesional y empresarial. La edición demo es gratuita, aunque está limitada en capacidad (almacenamiento y red). Las ediciones restantes están disponibles como software comercial.

II.5.3. SOPORTE DE TECNOLOGÍAS PARA DESARROLLO BASADO EN JAVA

La pila de tecnologías soportadas por la plataforma como servicio es una de las características distintivas entre proveedores. En este caso, la edición de Java soportada: Java SE y Java EE y el nivel de soporte a las características de Java EE, desde servlet/JSP hasta EJB pueden ser indirectamente determinadas considerando la variedad de marcos de trabajo y servidores de aplicaciones/contenedores de servlets soportados, respectivamente.

Otras características como el esfuerzo requerido para migrar aplicaciones Java heredadas (aplicaciones ya existentes) de un entorno *on-site* a un entorno en la nube es un aspecto importante especialmente en el caso de aplicaciones grandes de uso intensivo de datos [3]. Asimismo, el esfuerzo requerido para migrar una aplicación en la nube entre distintos proveedores de PaaS (portabilidad) es relevante en esta comparativa ya que se trata de una de las tendencias actuales en la investigación en computación en la nube. Estas características, aunque un tanto subjetivas, pueden ser evaluadas considerando la cantidad de "configuración extra" requerida en ambos casos, la cual puede variar desde rehacer la aplicación completamente hasta simplemente reubicar la aplicación sobre una nueva infraestructura sin necesidad de hacer cambios a nivel de código (en el caso de que no existan APIs específicas).

La figura siguiente presenta una comparativa del soporte de las siguientes características entre los proveedores de plataforma como servicio considerados en este documento: marcos de trabajo, servidores de aplicaciones/contenedores de servlets y Sistemas Gestores de Bases de Datos (DataBase Management Systems, DBMS) para desarrollo de aplicaciones Java, facilidad de migración de aplicaciones heredadas, portabilidad de aplicaciones en la nube y otras características complementarias.

PaaS	*Marcos de trabajo*	*Servidores/contenedores de servlets*	*DBMSs*	*Despliegue de aplicaciones sin marcos de trabajo especiales*	*Facilidad de migración de aplicaciones heredadas*	*Portabilidad de aplicaciones en la nube*	*Versión*
Amazon Elastic Beanstalk	No	Apache Tomcat	RDS (Relational Database Service) permite acceder a MySQL, Oracle, Microsoft SQL Server, PostgreSQL. DynamoDB provee acceso a bases de datos NoSQL"	Si	Si	Si	Producción
AppFog	Spring	Apache Tomcat	Redis, MongoDB, PostgreSQL, MySQL, RabbitMQ	Si	No	Si	Producción
Apprenda	No	Apache Tomcat	SQL Server, Oracle	Si	Si	Si	Producción (5.0)
Cloud Foundry	Spring, Lift	Apache Tomcat	MySQL, PostgreSQL. Redis, MongoDB, RabbitMQ	No	No	Si	Beta (v6)
CloudBees	Play, Lift	Apache Tomcat	MongoDB, CouchDB, Oracle, MySQL, SQL Server, PostgreSQL	Si	Si	Si	Producción
CloudControl	Spring, Play	Apache Tomcat, Jetty	MySQL, MongoDB, PostgreSQL	Si	No	Si	Producción
Cloudify	Play, Spring	Apache Tomcat, JBoss	MySQL, MongoDB	Si	Si	Si	Producción
CumuLogic	Spring	Apache Tomcat, JBoss, GlassFish	MySQL, Oracle, MongoDB	Si	Si	Si	Beta
DotCloud	Play	Apache Tomcat, Jetty	MySQL, MongoDB, PostgreSQL, Redis	Si	Si	Si	Producción
Engine Yard	No	Apache Tomcat, Jetty	MySQL, PostreSQL	Si	Si	Si	Producción
Google App Engine	Spring	Jetty	Cloud SQL, NoSQL, Cloud Storage	Si	No	Si	Producción
Heroku	Spring, Tapestry	Apache Tomcat, Jetty, Netty	MySQL, SQLite, PostgreSQL, MongoDB, CouchDB	Si	No	Si	Producción
Jelastic	Virtualmente cualquiera	Apache Tomcat, Jetty, GlassFish	MySQL, MongoDB, PostgreSQL, MariaDB, CouchDB	Si	Si	No (planeado)	Producción
Red Hat OpenShift	Play, Spring, Liferay	Apache Tomcat, JBoss, Jetty, GlassFish, WildFly	MySQL, MongoDB, PostgreSQL	Si	No	Si	Producción
Salesforce	Spring, Tapestry	Apache Tomcat, Jetty	Base de datos de la plataforma Salesforce1, Heroku Postgres	Si	No	Si	Beta
Stackato	Spring	Apache Tomcat, JBoss, Apache TomEE, Apache TomEE Plus	MySQL, MongoDB, PostgreSQL, Redis, ORACLE	No	No	No (Solo a nivel de datos)	Producción
Windows Azure	Spring	Vitualmente cualquiera a través de “worker roles”.	Windows Azure SQL, Windows Azure Table Storage (NoSQL) y bases de datos de terceros a través de “worker roles”.	No	No	No (Solo a nivel de datos)	Producción
WSO2 StratosLive	WSO2 Web Services Framework (WSF)	Apache Tomcat, JBoss, WSO2 Application Server	MySQL, PostgreSQL, H2, Servicio de almacenamiento de columnas WSO2	Si	No	Si	Producción

Figura 12. Soporte de tecnologías para desarrollo basado en Java.

II.5.3.1. Soporte al proceso de desarrollo y a la productividad del desarrollador

El grado de cobertura del ciclo de vida de desarrollo de aplicaciones Java es un factor relevante en la selección de un proveedor de plataforma como servicio entre distintas ofertas debido a que la cobertura completa del mismo puede suponer un ahorro en el tiempo, esfuerzo y presupuesto de proyectos al no requerir configuración de software e infraestructura de hardware subyacente. Finalmente, esto puede traducirse en un mayor retorno de inversión en el caso de proyectos con fines comerciales. Al mismo tiempo, la cobertura de más o menos todas las etapas del ciclo de vida puede suponer la inversión en innovación de los recursos ahorrados. Para fines prácticos, las etapas del ciclo de vida de aplicaciones Java consideradas en esta comparativa van desde desarrollo hasta mantenimiento, pasando por construcción (compilación), pruebas, despliegue y hospedaje.

Es importante destacar que en esta categoría también se incluyen características relacionadas con la productividad, es decir, las facilidades que permiten al desarrollador ahorrar tiempo y esfuerzo en el proceso de desarrollo y al mismo tiempo evitar errores y asegurar cierto grado de calidad. La existencia de Entornos de Desarrollo Integrado (Integrated Development Environments, IDEs) por encima de herramientas de línea de comandos es un ejemplo de las características consideradas en esta categoría.

La Figura 13. presenta una comparativa del soporte entre los proveedores de plataforma como servicio considerados en este documento de las siguientes características relacionadas con el soporte al proceso de desarrollo y a la productividad del desarrollador: integración de la etapa de pruebas, integración de la etapa de despliegue, integración de la etapa de construcción (compilación), soporte de: IDEs, herramientas de línea de comandos, consolas de administración Web, integración de servicios de terceros y herramientas de control de versiones y existencia de documentación y APIs, entre otras.

PaaS	*IDEs*	*Herramientas de línea de*	*Consola de administraci*	*Pruebas locales*	*Servicios de desarrollo/pr*	*Herramientas de control de versiones*	*Pruebas (en la nube)*	*Despliegue incremental*	*Registro de eventos*	*Documentación*	*API*	*Construcción (compilación)*
Amazon Elastic Beanstalk	Si	Si	Si	Si	No	Git	No	Si	Si	Buena	Si	No
AppFog	Si	Si	Si	Si	Si	Git. SVN, Mercurial	No	No	Si	Si	Si	Si
Apprenda	Si	Si	Si	Si	No	Git	Si	Si	Si	Si	Si	Si
Cloud Foundry	Si	Si	Si	Si	Si	Git	No	No	Si	Si	Si	No
CloudBees	Si	Si	Si	Si	Si	Bit, CVS	Si	Si	Si	Si	Si	Si
CloudControl	No	Si	Si	No	Si	Git, Bazaar	No	Si	Si	Si	Si	No
Cloudify	No	Si	Si	Si	Si	Git, SVN	Si	Si	No	Si	Si	No
CumuLogic	Si	Si	Si	Si	Si	Subversion, Git, Mercurial	Si	Si	Si	Si	Si	Si
DotCloud	No	Si	Si	Si	Si	Git,SVN, CVS	Si	Si	Si	Si	Si	Si
Engine Yard	Si	Si	Si	Si	No	Git	No	No	Si	Si	Si	Si
Google App Engine	Si	Si	Si	Si	No	Git, SVN	Si	Si	Si	Si	No	Si
Heroku	Si	Si	No	Si	No	Git server, Git client	Si	Si	Si	Si	Si	Si
Jelastic	Si	No	Si	Si	N/A	Git, SVN	Si (balance de carga)	Si	Si	Si	N/A	Si
Red Hat OpenShift	Si	Si	Si	Si	Si	Git	Si (balance de carga)	Si	Si	Si	Si	Si
Salesforce	Si	Si	Si	Si	Si	Git,SVN	Si	No	Si	Si	Si	Si
Stackato	Si	Si	No	Si	Si	Git	No	Si	No	Si	N/A	Si
Windows Azure	Si	Si	Si	Si	Si	Git, Mercurial, Team Foundation Server	Si (entorno de pruebas)	Si	Si	Si	Si	No
WSO2 StratosLive	Si	Si	No	Si	Si	Git, SVN	Si (automatización de pruebas)	Si	Si	Si	Si	Si

Figura 13. Soporte al proceso de desarrollo y a la productividad el desarrollador.

II.5.3.2. Características arquitectónicas

Una de las características más importantes de las PaaS es la escalabilidad. En este contexto, la escalabilidad se refiere a la habilidad de la plataforma de incrementar o disminuir la capacidad del servidor con el propósito de manejar el flujo de trabajo sin perder calidad del servicio ofrecido. Existen dos tipos de escalabilidad: vertical y horizontal. La primera de ellas se refiere al incremento de los recursos de cada nodo, tal como el tamaño de memoria RAM. La escalabilidad horizontal se refiere al incremento del número de nodos o instancias, reduciendo así las responsabilidades de cada uno de ellos.

Las PaaS han realizado grandes esfuerzos por proveer elasticidad en sus plataformas. En el contexto actual, elasticidad es entendida como la habilidad de un sistema para proveer o desproveer automáticamente recursos computacionales bajo demanda o cambios de carga de trabajo [4].

Por otra parte, una vez que son desplegadas las aplicaciones en la nube, es importante monitorear el desempeño de cada una de ellas para saber si se encuentran disponibles y respondiendo a las solicitudes que reciben. En la Figura 14 se presenta el análisis de las PaaS involucradas en el presente trabajo respecto a las características de escalabilidad, elasticidad, monitoreo web y tipo de nube.

PaaS	*Escalabilidad vertical*	*Escalabilidad horizontal*	*Elasticidad*	*Monitoreo Web*	*Tipo*
Amazon Elastic Beanstalk	Si	Si	Si	Si	Pública, Privada
AppFog	Si	Si	No	Si	Pública, Privada
Apprenda	Si	Si	Si	Si	Pública, Privada, Híbrida
Cloud Foundry	Si	Si	Si	Si	Privada, Pública
CloudBees	Si	Si	Si	Si	Pública, Privada, Híbrida
CloudControl	Si	Si	Si	Si	Privada
Cloudify	Si	No	Si	Si	Pública, Privada, Híbrida
CumuLogic	No	No	No	Si	Pública, Privada, Híbrida
DotCloud	Si	Si	Si	Si	Pública, Privada, Híbrida
Engine Yard	Si	Si	No	Si	Pública, Privada, Híbrida
Google App Engine	No	Si	Si	Si	Pública
Heroku	Si	Si	Si	Si	Pública, Privada, Híbrida
Jelastic	Si	Si	Si	Si	Pública, Privada
Red Hat OpenShift	No	Si	Si	Si	Pública, Privada, Híbrida
Salesforce	Si	Si	No	Si	Híbrida
Stackato	No	Si	Si	Si	Privada
Windows Azure	Si	Si	Si	Si	Pública, Privada, Híbrida
WSO2 StratosLive	No	Si	Si	Si	Pública, Privada, Híbrida

Figura 14. Características arquitectónicas.

II.5.3.3. Decisiones de negocio

Diversas características deben ser tomadas en cuenta al momento de elegir una PaaS para el despliegue de aplicaciones en la nube. Sin embargo, una de las más importantes es la inversión económica que la persona u organización pueda llevar a cabo para adquirir dicho servicio. En este contexto, existen PaaS que ofrecen planes gratuitos, y/o de paga. Dichos servicios varían entre otras cosas en la cantidad de recursos que son proporcionados por la PaaS para el despliegue de aplicaciones, entre estos recursos se encuentran la memoria RAM, el espacio de almacenamiento, la memoria caché, el ancho de banda entrante y saliente, entre otros. Resulta obvio, que los planes gratuitos ofrecen cada uno de los recursos antes mencionados en mucha menor proporción, y en caso de requerir de una mayor cantidad de recursos, el costo del servicio incrementa.

Otro factor importante a considerar es la disponibilidad de soporte por parte de la PaaS. La calidad del soporte va ligada en la mayoría de los casos con el plan (gratuito o de paga) que se adquiera. Dicho soporte va desde foros y comunidades, hasta soporte técnico especializado durante las 24 horas del día.

Muchas de las organizaciones que deciden crear o migrar sus aplicaciones a la nube, analizan de manera detallada el nivel de seguridad que proporciona la PaaS, ya que la información contenida o circulante en dichas aplicaciones demanda altos niveles de confidencialidad.

En la Figura 15 se presenta un análisis de cada uno de los factores antes mencionados con respecto a las plataformas analizadas en el presente trabajo.

PaaS	*Servicio gratuito*	*Precio (Dólares)*	*Soporte*	*Seguridad*	*Código abierto*
Amazon Elastic Beanstalk	Si	$35,115 mensual	Plan gratuito: Foros Plan de paga: Soporte AWS (Básico, Desarrollador, Negocios, Empresarial) costo adicional	HTTPS, SSH, ACL, Firewall, VPC (Virtual Private Cloud), SSL	No
AppFog	No	$20 - $720 mensual	Documentación, correo electrónico	SSL	No
Apprenda	No (Solo periodo de prueba)	No especificado	Soporte de la comunidad, correo electrónico, teléfono, soporte en línea	SSL, HTTPS	No
Cloud Foundry	Si	$0.03 /Gb RAM/Hora	Blog, Wiki, Documentación	Servicio de autenticación de usuarios (UAA)	Si
CloudBees	Si	$60 - $200 mensual	Foros, soporte técnico	SSL	No
CloudControl	Si	$29-$499 mensual	Plan gratuito: Soporte de la comunidad Plan de paga: soporte técnico profesional	SSL	No
Cloudify	Si	No especificado	Teléfono, correo electrónico, foros	SSL	Si
CumuLogic	Si	Gratuito	Teléfono, correo electrónico, foros	SSL	No
DotCloud	Si	$47.52 mensual aprox.	Soporte en línea	SSL	Si
Engine Yard	No (para Java no disponible)	$40-$1060 mensual	Planes de soporte técnico (Estándar, Premium, Managed) (costo adicional)	SSL, SSH	Si
Google App Engine	Si	Almacenamiento: $0.18 Gb/mes Memoria caché: $0.12 Gb/hora SSL: $39 mensual Premier $150 mensual	Foros	SSL	No
Heroku	Si	$0.05-$0.80 hora	Correo electrónico y teléfono	SSL	No
Jelastic	No (Solo periodo de prueba)	$0.027 hora	Correo electrónico	SSL	No
Red Hat OpenShift	Si	$20 mensual	Plan gratuito: Correo electrónico, soporte en línea, foros Plan de paga: Servicio de soporte global atención personalizada y por teléfono y al portal del cliente	SSL	Si
Salesforce	No (Solo periodo de prueba)	$25 mensual	Plan gratuito: Foros, blogs, Plan de paga: Soporte de atención telefónica, servicios de entrenamiento, servicio al portal del cliente	SSL	No
Stackato	Si	No especificado	Plan gratuito: preguntas frecuentes y foros Plan de paga: correo electrónico, teléfono	Docker, SSL, AppArmor, SSH, SCP	No

PaaS	*Servicio gratuito*	*Precio (Dólares)*	*Soporte*	*Seguridad*	*Código abierto*
Windows Azure	No (Solo periodo de prueba)	$0.027 hora	Preguntas frecuentes, foros Soporte de desarrollador: correo electrónico y teléfono	Desconocido	No
WSO2 StratosLive	Si	No especificado	Plan gratuito: foros Plan de paga: Soporte integrado (evaluación, entrenamiento, desarrollo), asistencia telefónica y al portal del cliente	HTTPS, DMZ, Amazon Cloud Security Gateway, Java Security Manager	Si

Figura 15. Decisiones de negocio.

II.6. PLATAFORMAS MÓVILES

Este apartado muestra un breve estudio de las plataformas móviles existentes en el año 2013 y las tendencias. Aunque esta imagen ha cambiado en los últimos 2 años, este trabajo se ha desarrollado entre los años 2011 y 2014, siendo en el 2013 donde se obtuvo el primer prototipo, por lo que se ha decido mantener la información existente en ese año 2013 en vez de actualizar los datos a los actuaes en 2015, pues eso ayuda a comprender ciertas decisiones tomadas durante el desarrollo de este trabajo.

En el año 2013, el diseño de aplicaciones para plataformas móviles estaba teniendo una gran importancia en todas las organizaciones. En los últimos años, hubo una gran lucha por distintas plataformas por hacerse con la cuota más importante en el mercado y por lo tanto ha habido una gran proliferación de distintas plataformas móviles. En este apartado se realiza una breve comparación para así detectar qué plataforma móvil era la más adecuada entonces para el desarrollo de la aplicación móvil a desarrollar en esta tesis doctoral.

	Apple iOS 7	Android 4.3	Windows Phone 8	BlackBerry OS 7	Symbian 9.5
Compañía	Apple	Open Handset Alliance	Microsoft	RIM	Symbian Foundation
Núcleo del SO	Mac OS X	Linux	Windows NT	Mobile OS	Mobile OS
Licencia de *software*	Propietaria	Software libre y abierto	Propietaria	Propietaria	Software libre
Año de lanzamiento	2007	2008	2010	2003	1997
Fabricante único	Sí	No	No	Sí	No
Variedad de dispositivos	modelo único	muy alta	media	baja	muy alta
Soporte memoria externa	No	Sí	Sí	Sí	Sí
Motor del navegador web	WebKit	WebKit	Pocket Internet Explorer	WebKit	WebKit
Soporte Flash	No	Sí	No	Si	Sí
HTML5	Sí	Sí	Sí	Sí	No
Tienda de aplicaciones	App Store	Google Play	Windows Marketplace	BlackBerry App World	Ovi Store
Número de aplicaciones	825.000	850.000	160.000	100.000	70.000
Coste publicar	$99 / año	$25 una vez	$99 / año	sin coste	$1 una vez
Actualizaciones automáticas del S.O.	Sí	depende del fabricante	depende del fabricante	Sí	Sí
Familia CPU soportada	ARM	ARM, MIPS, Power, x86	ARM	ARM	ARM
Máquina virtual	No	Dalvik	.net	Java	No
Aplicaciones nativas	Siempre	Sí	Sí	No	Siempre
Lenguaje de programación	Objective-C, C++	Java, C++	C#, muchos	Java	C++
Plataforma de desarrollo	Mac	Windows, Mac, Linux	Windows	Windows, Mac	Windows, Mac, Linux

Figura 16. Comparativa de las principales plataformas móviles.

De ese estudio destacaron cinco plataformas móviles que dominan el mercado internacional: Apple IOS, Android, Microsoft mobile, Blackberry OS y Simbian. En la Figura 16 se muestra una comparación de las principales características de estas plataformas[1]. Como se puede observar las plataformas Android y Blackberry OS proporcionaban características interesantes en esta tesis doctoral como el desarrollo de aplicaciones en HTML5 y en Java.

Otro aspecto fundamental necesario para decidir qué plataforma puede ser la más adecuada es su cuota de mercado. En la Figura 17 se puede observar un estudio realizado por

[1] Universidad Politécnica de Valencia. Las plataformas para móviles en 2013 @UPV. Accesible en http://www.youtube.com/watch?v=N6NlbpN1zVk

la empresa Gartner Group[2], donde se muestra la evolución del mercado de los sistemas operativos para móviles según el número de terminales vendidos. Se puede destacar el espectacular ascenso de la plataforma Android, que le ha permitido alcanzar en unos años una cuota de mercado superior al 75%.

Como se puede observar en la Figura 17, Blackberry OS representa la cuarta plataforma móvil más importante y Android es la primera con mucha diferencia.

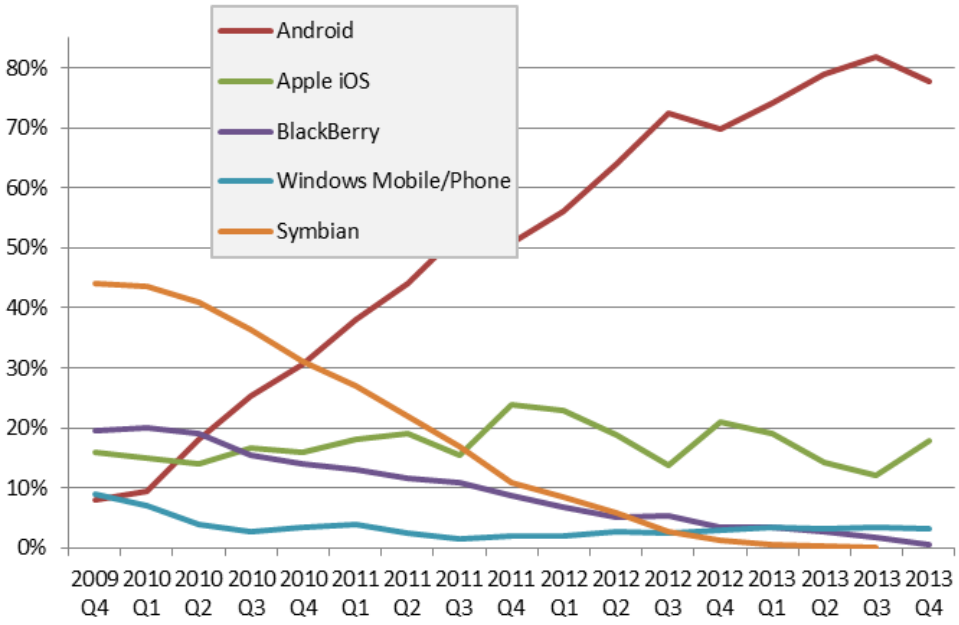


Figura 17. Porcentaje de teléfonos inteligentes vendidos según su sistema operativo en el mundo (fuente: Gartner Group).

[2] Gartner Group 2013, http://www.gartner.com/newsroom/id/2573415

Sistema operativo	Unidades 2Q13	Cuota de Mercado 2Q13 (%)	Unidades 2Q12	Cuota de mercado 2Q12 (%)
Android	177,898.2	79.0	98,664.0	64.2
iOS	31,899.7	14.2	28,935.0	18.8
Microsoft	7,407.6	3.3	4,039.1	2.6
BlackBerry	6,180.0	2.7	7,991.2	5.2
Bada	838.2	0.4	4,208.8	2.7
Symbian	630.8	0.3	9,071.5	5.9
Others	471.7	0.2	863.3	0.6
Total	225,326.2	100.0	153,772.9	100.0

Figura 18. Ventas en el mundo a usuarios finales de Smartphones por sistema operativo en 2013 (Millares de unidades) (fuente: Gartner Group)

De este estudio, junto con las relaciones existentes entre el autor de este trabajo con la empresa RIM (propietarios del sistema Blackberry), fueron los que hicieron que se tomase la decisión de desarrolloar la aplicación móvil de comunicación inicialmente en Android y Blackberry.

Capítulo III. ARQUITECTURA PROPUESTA

III.1. INTRODUCCIÓN

En este capítulo se detalla la arquitectura funcional del sistema desarrollado en esta tesis. Se describirán los módulos de los que está formado el mismo en base a las reuniones y estudios realizados.

El objetivo de este apartado es realizar una primera aproximación sobre la estructura funcional del sistema para explicar brevemente los módulos en los que el sistema se divide para desarrollar detenidamente cada uno de ellos en diferentes apartados de este capítulo.

Esta tesis plantea el desarrollo de una interfaz de diseño para todos, que permita el manejo y la interacción con un dispositivo móvil a través de órdenes de voz en lenguaje natural. En resumen, el trabajo realizado ha consistido en desarrollar y mejorar una tecnología que permita a los sistemas informáticos "entender" el lenguaje coloquial, es decir, el lenguaje usado por las personas para expresarse en su entorno. Estas tecnologías pueden ser aplicadas en este trabajo en la interacción con los dispositivos móviles y toda la funcionalidad que permiten hoy día.

La consecución de este trabajo favorece la adaptación de estos dispositivos móviles a personas discapacitadas para que puedan utilizar toda la funcionalidad que proporcionan de manera sencilla y cómoda por este colectivo. Así, otro de los objetivos sociales es desarrollar tecnologías para la ayuda a este colectivo que les permita solventar las dificultades que tienen en el día a día y gracias de un módulo de reconocimiento del habla en lenguaje natural implantado en estos dispositivos puede resultar de una ayuda inestimable para conseguir una mayor integración de estos colectivos en la sociedad actual.

III.1.1. ARQUITECTURA FUNCIONAL

La arquitectura funcional del sistema planteado es una arquitectura cliente-servidor basada en tecnología de *cloud-computing*. Los terminales móviles cuentan con una aplicación que será la encargada de captar el sonido/voz y de enviar esa voz a un servidor que se encarga de convertir la voz en texto (reconocimiento automático de voz o ASR) y

posteriormente envía ese texto a una aplicación de servidor, donde se encuentran implementadas diversas tecnologías basadas en Web Semántica y Procesamiento de Lenguaje Natural, con el fin de obtener la semántica de la orden y generar un comando estructurado que posteriormente la aplicación instalada en el terminal móvil es capaz de procesar. Esta opción de que tanto el reconocimiento de voz como la interpretación semántica se hagan en servidores y no en el propio terminal, permite que se puedan utilizar grandes librerías (diccionarios, listas de la palabras, analizadores morfo-sintácticos,...) que consumen grandes recursos computacionales todavía no disponibles en la mayoría de móviles o que hace que el consumo de batería se incremente en aquellos *smartphones* que si disponen de esos los recursos suficientes, lo que en definitiva dota al sistema de una alta capacidad de poder prestar los nuevos usos buscados en los objetivos de la presente tesis doctoral. Además, esta arquitectura está en harmonía con posibles fines comerciales que puedan resultar de este trabajo y que consisten en que las operadoras puedan ofrecer esta tecnología como un servicio a sus clientes.

El uso de tecnologías de *cloud-computing* no se debe únicamente a que sea una tecnología de moda, ya que se prevé que un servicio de estas características crezca en número de usuarios, ya sea porque se adapta a nuevos idiomas, porque más operadoras lo requieran o porque el número de usuarios de telefonía móvil que contratan un plan de datos incrementa exponencialmente. La “nube” como se conoce habitualmente, favorece en gran medida la escalabilidad de este tipo de servicios, pudiendo, de una manera transparente tanto para el consumidor (cliente de telefonía), para el cliente (operador de telefonía), como para los proveedores del servicio, aumentar los recursos necesarios para prestar el servicio de una manera sencilla y rápida, sin necesidad de invertir grandes cantidades en nuevas infraestructuras informática y de telecomunicaciones.

En la siguiente figura se muestra un diagrama con la arquitectura inicial propuesta para el desarrollo del trabahi en función de los detalles expuestos anteriormente:

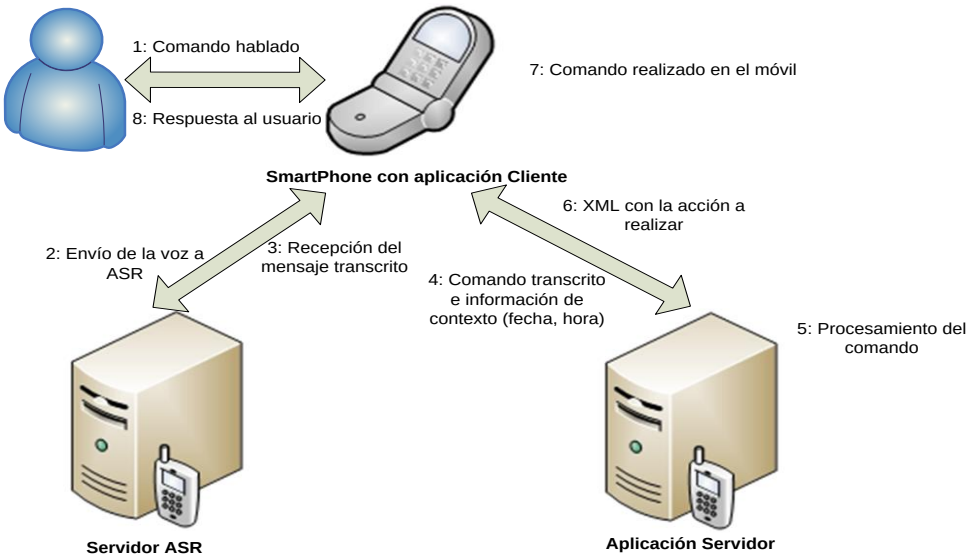


Figura 19. Diseño global del sistema propuesto.

Como se puede apreciar, y como se mencionaba anteriormente, la arquitectura plateada permitirá al usuario, simplemente pulsando un botón de su terminal móvil, iniciar la aplicación que permitirá captar la voz del usuario para que ésta sea transcrita usando un servicio instalado en un servidor ASR para la conversión de voz a texto. Una vez realizada dicha conversión, el texto reconocido será enviado al servicio proporcionado por la aplicación del servidor con el fin de analizar el significado de ese texto con el comando expresado en lenguaje natural para crear un comando estructurado XML que será fácilmente procesable por la aplicación instalada en el terminal móvil, realizando la acción deseada por el usuario. Veamos un ejemplo que ayude a esclarecer cualquier duda que pueda surgir sobre la aplicación y la arquitectura:

1. El usuario pulsa un botón o tecla de acceso rápido que ejecuta el cliente instalado en el smartphone y a continuación dice un comando verbal (p.e. “apúntame una reunión el próximo lunes con Pedro García” [voz]).
2. La aplicación instalada en el móvil y resultado de esta tesis doctoral enviará el audio al servidor ASR, capaz de transcribir la voz en texto.
3. Una vez procesado este texto, el servidor ASR enviará a la aplicación móvil el texto transcrito (siguiendo el ejemplo: “apúntame una reunión el próximo lunes con Pedro García” [texto]).
4. Ese texto será ahora enviado por la aplicación del móvil al servicio proporcionado por el sistema NLP ofrecido como servicio HTTP a través de la red.

5. La aplicación del servidor NLP se encargará de procesar el comando y obtener su significado utilizando las tecnologías desarrolladas en esta tesis.
6. Una vez identificado ese comando se creará un comando estructurado en XML con la información legible para la aplicación móvil:

```
<xml...>
      <AddCalendar...>
            <startDay>31/10/2011</startDay>
            <guest>Pedro García</guest>
      </AddCalendar>
</xml>
```

7. Una vez recibido el comando estructurado, la aplicación del móvil sabrá qué cambios tiene que reflejar en el móvil, creando un nuevo evento en la agenda del mismo para el día 31/10/2010 e indicando que la reunión se celebrará con Pedro García.

A continuación se describe de manera detallada cada una de las arquitecturas de los módulos diferenciados: subsistema ASR, subsistema NLP y aplicación cliente.

III.1.1.1. Subsistema NLP

El objetivo de esta aplicación es proporcionar un servicio al que se le escriba un comando en lenguaje natural escrito y devuelva el comando estructurado en un formato XML que se explica más adelante en este capítulo. Para ello esta aplicación debe implementar un motor de procesamiento del lenguaje natural (NLP) para el análisis de comandos para dispositivos móviles. En la siguiente figura podemos ver la arquitectura funcional de este módulo.

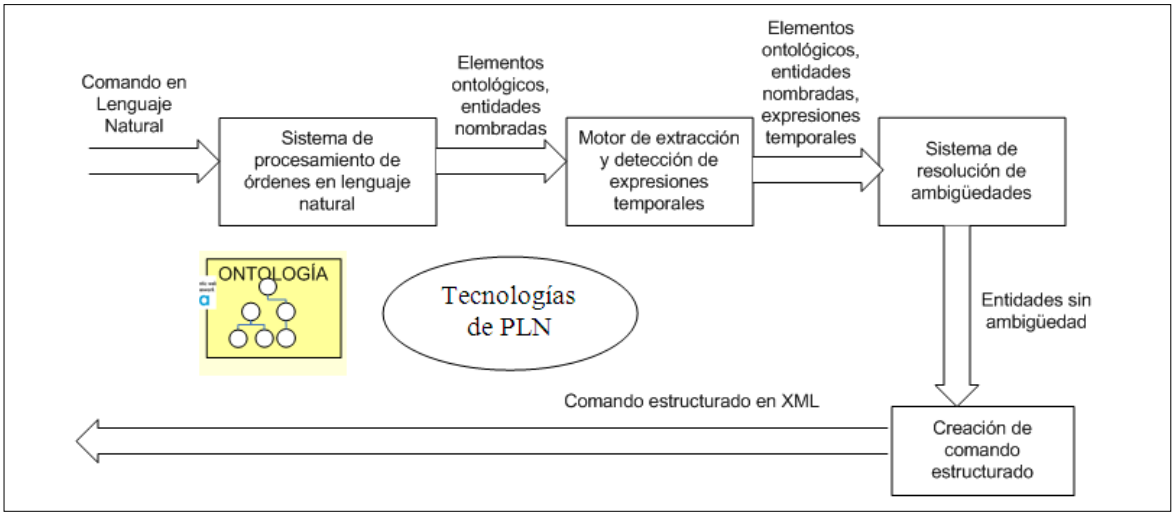


Figura 20. Arquitectura funcional del motor de NLP para el análisis de comandos.

A continuación se describen detalladamente cada uno de los submódulos.

III.1.1.1.1. Sistema de procesamiento de órdenes en lenguaje natural

Este submódulo utiliza para procesar el comando en lenguaje natural, una ontología del dominio que representará las posibles órdenes a ejecutar en un terminal móvil y un conjunto de recursos lingüísticos de procesamiento del lenguaje natural. La salida de este módulo es un conjunto de anotaciones (elementos ontológicos y entidades nombradas) que identifican entre otras cosas la orden dada, la aplicación donde se ejecutará la orden, así como alguna entidad nombrada y conocimiento dentro del comando como puede ser un teléfono, correo electrónico o un nombre de persona.

III.1.1.1.2. Motor de extracción y detección de expresiones temporales

En este módulo se utiliza estándar TIMEX2 (http://fofoca.mitre.org/) para la representación de expresiones temporales. Este sistema es capaz de detectar, anotar y representar anotaciones temporales usando reglas JAPE que son apropiadas para detectar, y posteriormente anotar, expresiones temporales escritas como “el lunes de la semana que viene”, “a las 6 y media de la tarde”, “entre las 10 y las 12 de la noche”, etc. Posterior a la anotación, se realiza un proceso de representar la expresión temporal usando el estándar TIMEX2.

Este sistema se basa en reglas lógicas capaces de resolver complejas y ambiguas expresiones temporales (p.e. “el primer lunes del mes que viene” se refiere al 2/8/2010). Para ello se creará un sistema de razonamiento basado en Java y que usará un motor de inferencia basado en reglas, similar a JRules. También se hará especial hincapié en el estudio de los casos concretos de ambigüedad en base al conjunto de expresiones aceptado por el sistema determinando reglas que permitan resolver la ambigüedad.

III.1.1.1.3. Sistema de resolución de ambigüedades

El lenguaje es por sí ambiguo por eso es muy importante poder detectar las posibles ambigüedades surgidas en las fases anteriores para poder resolverlas en base a heurísticas.

Para poder diseñar este sistema se ha hecho un estudio lingüístico para determinar posibles ambigüedades del lenguaje en el dominio de esta tesis doctoral para, una vez detectadas, generar unas reglas lógicas que permitan desambiguar los comandos

procesados. Este módulo trata de eliminar posibles ambigüedades derivadas de la naturaleza del lenguaje que puedan confundir al sistema en la detección del comando a realizar. Por ejemplo, el comando "recuérdame que tengo que enviar un correo electrónico a Antonio diciéndole que ya está reservado el hotel para las vacaciones" está asociado a la creación de una nueva tarea en la aplicación del móvil, no al envío de un correo electrónico.

III.1.1.1.4. Creación de comando estructurado

Como resultado del procesamiento de las fases anteriores el comando de voz se habrá descompuesto en las entidades que lo forman identificadas de manera unívoca como un conjunto de anotaciones sobre el comando original. En base a estas entidades el sistema aplica un conjunto de reglas de transformación que permiten construir un comando estructurado en XML conteniendo toda la información relevante.

Estas transformaciones se aplican sobre las anotaciones que las fases anteriores han generado en base al esquema XML que determina la composición de cada uno de los comandos posibles generando el comando estructurado de salida.

III.1.1.2. Subsistema ASR

El servidor ASR se encarga exclusivamente de la transcripción de voz a texto. En el caso de los sistemas de reconocimiento de voz, existen dos alternativas que se estudiarán utilizando la que obtenga mejores resultados o una conjunción de ambas.

Por un lado la utilización de un reconocimiento basado en comandos que se apoyará en una gramática definida que modele el conjunto de expresiones que pueden utilizarse para dar los comandos aceptados por el sistema.

Por otro lado la utilización de un reconocimiento basado en transcripción de discurso libre. En este caso, y en base a unos modelos fonéticos y probabilísticos así como a un determinado diccionario, se intentará obtener una transcripción de todo el texto literal que conforma el comando, de manera similar a un proceso de dictado puro.

El reconocimiento basado en comandos nos permite identificar el tipo de comando concreto de una manera más precisa así como los distintos componentes del mismo. El reconocimiento basado en dictado es el que posibilita el reconocimiento de los componentes más abiertos de cada comando, como el cuerpo de un mensaje o de un correo electrónico.

En la figura siguiente puede apreciarse la arquitectura funcional de lo que sería la combinación de ambas posibilidades. En este caso existe un componente llamado *Text Builder* que se encarga de dar la salida final en base a las distintas hipótesis que el ASR Dictado puede generar, al comando elegido por el ASR Comando y a un conjunto de reglas que determinan la salida en base a las distintas posibilidades que pueden generarse.

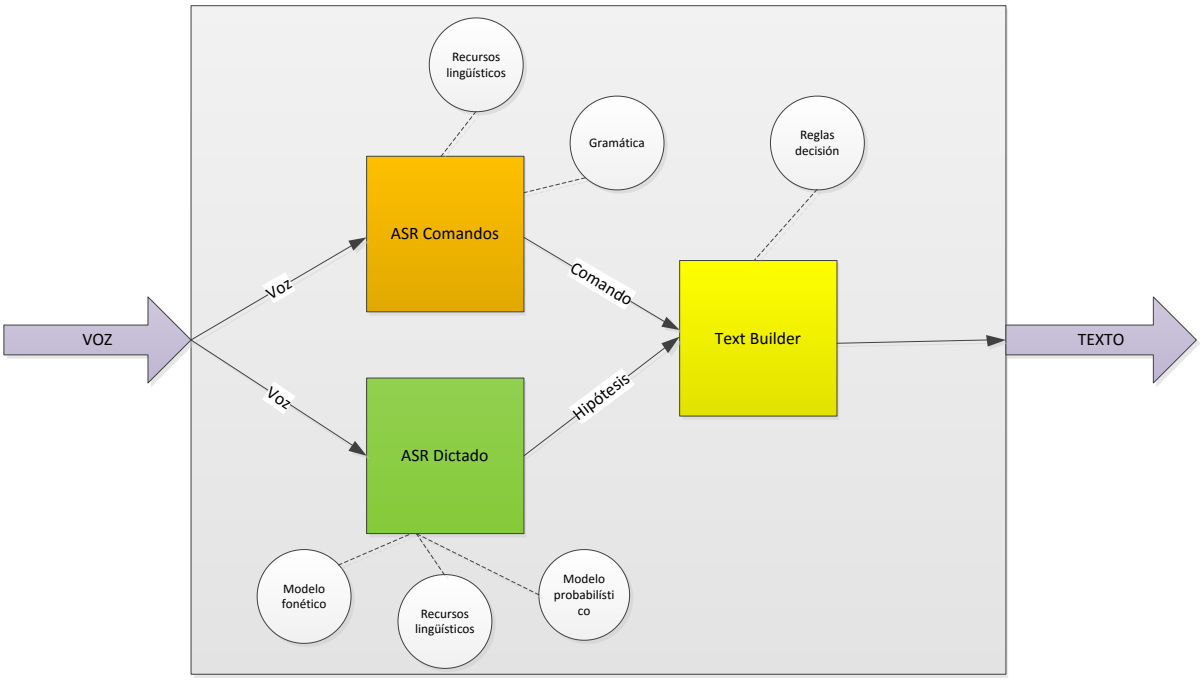


Figura 21. Arquitectura funcional servidor ASR.

III.1.1.3. Aplicación cliente

La aplicación cliente que se ejecuta en el terminal móvil es la herramienta de acceso al sistema y se encarga tanto de enviar el comando de voz del usuario al servidor para su reconocimiento como de recoger el comando estructurado resultante y ejecutar en el terminal la acción correspondiente.

La implementación de la aplicación cliente variará entre las distintas plataformas móviles existentes aunque su arquitectura funcional será la misma en todos los casos.

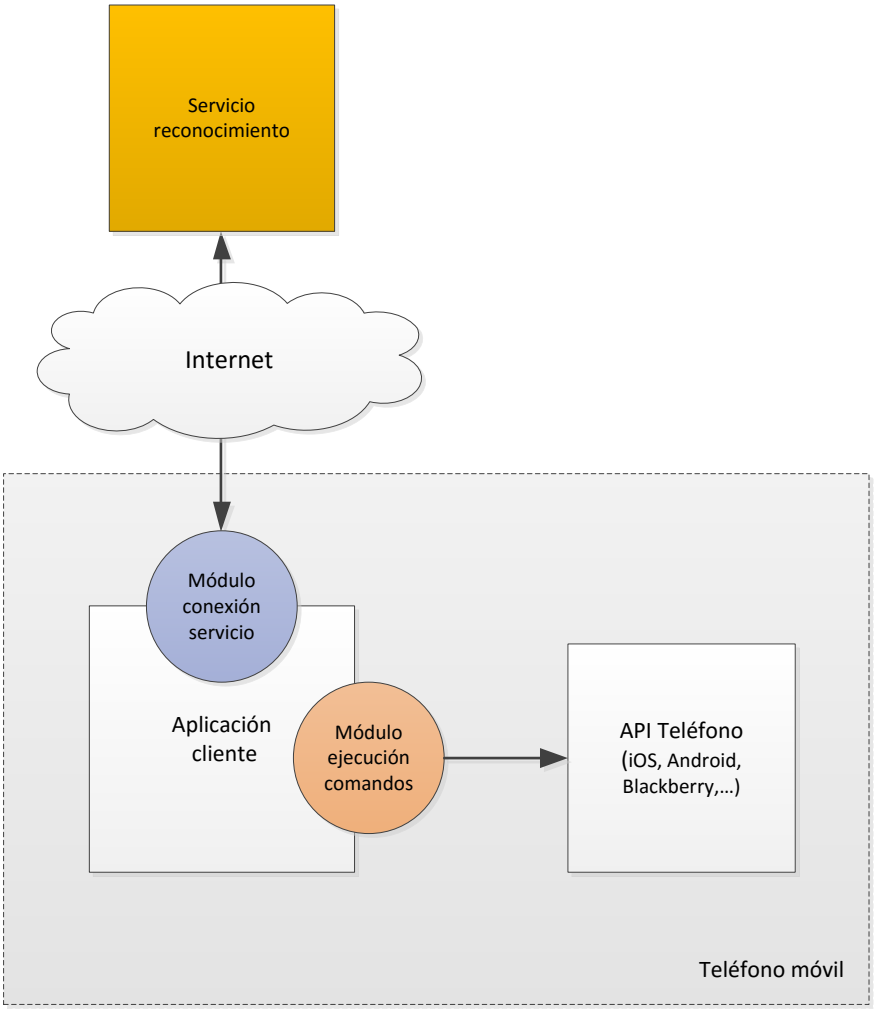


Figura 22. Arquitectura funcional cliente móvil.

Como puede apreciarse en la figura existen dos módulos funcionales equivalentes en todos los clientes sea cual sea la plataforma base. El módulo de conexión con el servicio contiene la lógica capaz de enviar el audio del comando al servidor ASR y de recibir el comando estructurado de respuesta. El módulo de ejecución de comandos será el encargado de actuar sobre el teléfono para ejecutar el comando recibido.

Ambos módulos tendrán una definición idéntica en todas las plataformas, variando la implementación en lo que se refiere a la interacción con la plataforma móvil.

El resto de la aplicación cliente está compuesta básicamente por la interfaz de usuario, absolutamente dependiente y diferente, incluso a nivel conceptual, en cada una de las plataformas.

III.1.2. ARQUITECTURA CLOUD-COMPUTING

La computación en la nube representa un nuevo paradigma dentro del mundo de la programación y de las Tecnologías de la Información. Este paradigma define un modelo de computación basado en Internet en el cual se comparten los recursos, el software y la información, distribuyéndola en ordenadores o dispositivos bajo demanda. Los usuarios finales no necesitan conocer los detalles de la localización y configuración del sistema remoto que les presta el servicio.

La justificación del despliegue de esta arquitectura definida en la nube viene por las ventajas de escalabilidad y flexibilidad que este paradigma aporta al despliegue de aplicaciones. Estas características se tuvieron en cuenta para adaptar la solución propuesta al nuevo entorno de computación.

Este documento recoge, en cierta parte, todo el proceso realizado para llevar a cabo la migración de la la solución diseñada a un modelo en la nube. Además, al final del documento se ofrece un estudio de rendimiento realizado sobre la plataforma desplegada.

III.1.3. DESPLIEGUE EN LA NUBE

La arquitectura y diseño del trabajo tuvo en cuenta desde los inicios la posibilidad de que el servicio pudiera desplegarse en una infraestructura de nube.

Las características del servicio actual hacen que no sea necesario establecer sesión con el cliente (dispositivo móvil) lo que permite un balanceo y distribución de carga entre distintas instancias petición a petición, lo que le confiere una enorme escalabilidad.

A su vez, los recursos lingüísticos y modelos empleados en el procesamiento del texto son fácilmente replicables entre las distintas instancias del servicio que pudieran existir en una infraestructura de nube pues no son de gran tamaño ni se ven afectados por el funcionamiento del servicio, evitando necesidad de sincronización entre las distintas instancias. Además estos pueden variarse en caliente, por lo que podrían ser actualizados sin necesidad de detener el servicio.

Para este fin se ha creado un módulo con interfaces configurables para distintos servicios de nube que es capaz de obtener las instancias activas en un momento determinado y actualizar los recursos así como la imagen de la instancia base de manera que los cambios sean permanentes.

En el paradigma de computación en nube existen tres modelos básicos de despliegue: modelo de despliegue privado, público o hibrido y cada modelo de despliegue aporta características dispares que a continuación analizamos de manera individual:

- Modelo de despliegue privado. Este modelo ofrece una mejora con las operaciones de gestión relacionadas con la gestión de información debido a que los datos se ubican en servidores locales a la organización pero, por otro lado, ofrece otras características como escalabilidad y flexibilidad limitadas, gestión limitada de picos de demanda, es decir, esta solución estaría destinada para organizaciones que necesitan securizar sus datos porque manejan información de carácter sensible y que no requieren flexibilidad en sus aplicaciones. Estas características no se ajustan al tipo de sistema desarrollado dado que, por un lado se quiere que el sistema sea lo más flexible posible, que este capacitado para abarcar cuanto mas usuarios mejor y que la información que almacena no es de carácter sensible por lo que este modelo queda descartado.
- Modelo de despliegue público: Este modelo es justamente lo opuesto al anterior, en lo que escalabilidad y flexibilidad se refiere, una de las contraindicaciones por el que este modelo no es instaurado en muchos casos en las organizaciones es debido a la incertidumbre sobre la seguridad de la información, ya que toda la información es publicada en Internet y no todas las organizaciones están tranquilas con esta idea. Pero para el sistema desarrollado, en concreto, donde no existe ningún tipo de sensibilidad con la información que los usuarios van a generar, este modelo de despliegue se convierte en el idóneo debido a la búsqueda de escalabilidad que la plataforma requiere.
- Modelo híbrido: Proporciona una solución que sigue el modelo privado en cuanto a privacidad de los datos y el modelo público de despliegue en cuanto a escalabilidad y flexibilidad se refiere. Sin duda alguna, este modelo de despliegue sería el más interesante a tener en cuenta si algún día la aplicación móvil trabajase con algún tipo de información del usuario de carácter sensible.

Para la configuración del modelo público, en primer lugar debe seleccionarse un proveedor que se adapte al software implementado y que proporcione los servicios idóneos para el despliegue de la arquitectura. Como se ha analizado durante los primeros apartados del documento, existen diferentes proveedores PaaS (plataformas

como servicio) que tienen altas capacidades para ejecutar los servicios de la aplicación. Entre todos los analizados se seleccionó el proveedor CloudBees. Uno de los aspectos más relevantes de porque fue seleccionado este proveedor es debido a que soporta el modelo hibrido de despliegue, que como se puede ver en las tablas, no todos los proveedores disponen de esta característica. Además, entre todos los proveedores que implementan este modelo, es uno de los proveedores que mejor características arquitectónicas ofrece y uno de los que más soporte tecnológico proporciona. Una vez seleccionado CloudBees como proveedor PaaS, el siguiente paso fue decidir la arquitectura que va a ser desplegada en el modelo público. La Figura 23 representa gráficamente la configuración del despliegue.

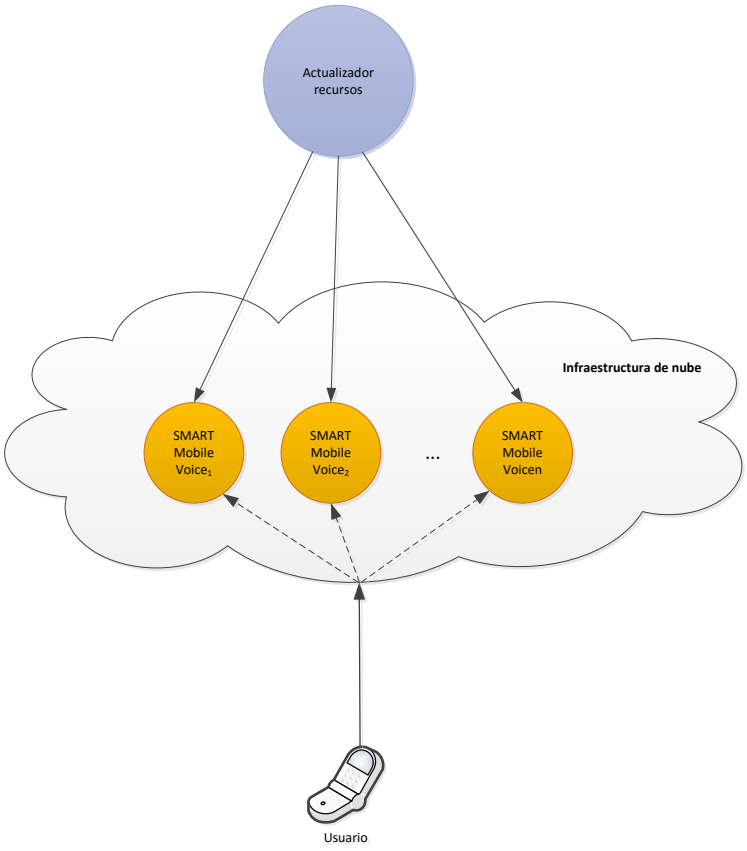


Figura 23. Arquitectura del servicio desplegado en la nube.

Si se analiza detalladamente la Figura 23, la arquitectura en la nube constará de un único servidor físico o virtual donde serán creadas tantas instancias de la aplicación como tipos de usuarios de la aplicación existan. La Figura 24, muestra una especificación de la Figura 23, en ella se pueden ver como se gestionaran las instancias de la aplicación dentro de la nube.

Actualmente, existen diversas estrategias de despliegue de soluciones multi-tenant (múltiples clientes) que permiten la distribución, aislamiento y personalización de los recursos que la aplicación requiere en la nube. La solución que ha sido seleccionada es la aplicación con múltiples instancias que permite dar soporte a cada cliente con su aplicación de instancia dedicada en el hardware compartido que puede ser un sistema operativo o el servidor de middleware en el entorno del proveedor. Es decir, el proveedor proporciona un sistema operativo sobre el cual se crean tantas instancias como perfiles de clientes usen la aplicación. Esta solución permite ofrecer un servicio más personalizado.

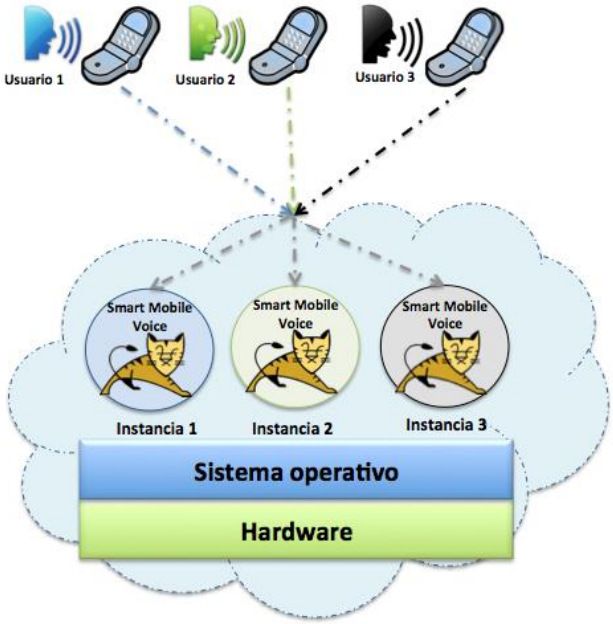


Figura 24. Arquitectura desplegada en la nube.

La Figura 24 muestra gráficamente la arquitectura desplegada en la nube, en ella se pueden ver como usuarios con diferentes perfiles ejecutarán diferentes instancias de la misma aplicación sobre un mismo sistema operativo. La finalidad de esta arquitectura es, como se ha mencionado anteriormente, conseguir para cada usuario un aislamiento y personalización del servicio.

III.1.4. ONTOLOGÍA

En este apartado se detalla la ontología que representa los posibles comandos, aplicaciones y entidades nombradas más importantes dentro del dominio de aplicación, esto es, el control de los dispositivos móviles mediante lenguaje natural.

III.1.4.1. Ontología en la aplicación

En este apartado se explica la ontología desde el dominio de la aplicación describiendo las principales taxonomías y clases de la misma.

III.1.4.2. Descripción de la Ontología.

El modelo ontológico u ontología que representa el conocimiento incluido en un comando en lenguaje natural es un fichero OWL que representa el comando realizada con toda la información semántica que ha sido posible extraer de ella. La finalidad es describir el comando de manera que otros módulos puedan utilizar la información contenida en él para poder interpretarlo y ejecutarlo.

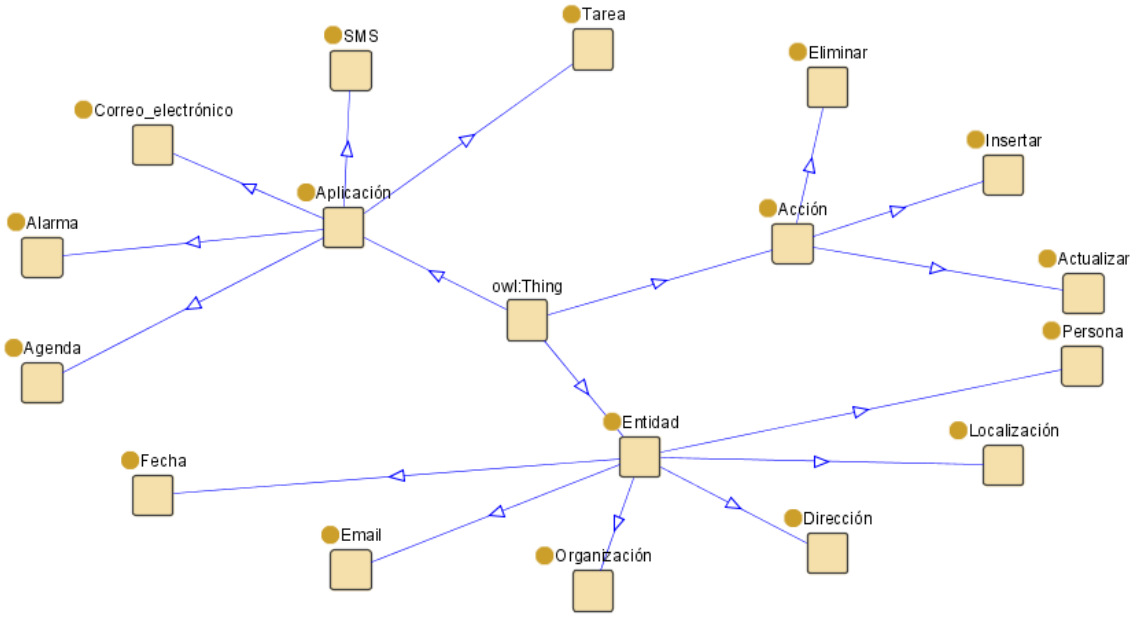


Figura 25. Ontología que representa el modelo del comando

Esta ontología se compone de 3 clases principales (ver Figura 25):

- **Aplicación**: Representa la aplicación software o programa sobre la cual va a recaer la acción. Este puede ser la agenda, alarma, calendario, correo electrónico o SMS.
- **Acción**: Representa la acción a realizar que puede ser insertar, modificar o eliminar.

- **Elemento**: Representa entidades que pueden formar parte de un comando determinado como personas, email, números de teléfono, direcciones, fechas, horas, etc.

Esta tesis doctoral va a utilizar para representar sus ontologías el lenguaje OWL en su versión OWL DL. Estas ontologías modelan el dominio en base a cinco elementos ontológicos:

- Classes: Las clases son los conceptos del dominio. Serían como un equivalente a las clases en programación orientada a objetos (POO).
- DatatypeProperties: Los datatypeProperties son los atributos de los conceptos que pueden tomar valores simples o complejos.
- ObjectProperties: Los objectProperties van a representar las relaciones no taxonómicas existentes entre los conceptos de la ontología. Se pueden definir restricciones sobre ellas.
- Individuals: Los individuals son las instancias. Serían el equivalente a los objetos en POO. Una instancia puede verse como un concepto del dominio en el que todos sus atributos y relaciones están rellenas, en definitiva serán siempre las hojas de los árboles taxonómicos definidos en la ontología.
- Subclass of relationships: Las relaciones taxonómicas son relaciones IS_A o de herencia y estas tienen un tratamiento distinto a los ObjectProperties. Se definen como rdfs:subclassOf y no como ObjectProperties.

III.1.4.3. Recursos lingüísticos para detección de comandos verbales

Para obtener el modelo del comando se han desarrollado diversos recursos lingüísticos que permitan detectar acciones, dispositivos o aplicaciones donde realizar esas acciones y la detección de entidades nombradas como son nombres propios, lugares, direcciones, teléfonos, direcciones de correo electrónico, etc.

Dichos recursos se han implementado utilizando la herramienta para el desarrollo de aplicaciones de minería de texto GATE y más concretamente de sus Gazzeteers y reglas JAPE para poder localizar estas entidades. En la Figura 26 se muestra un ejemplo de la deteción de estos recursos dentro de un comando.

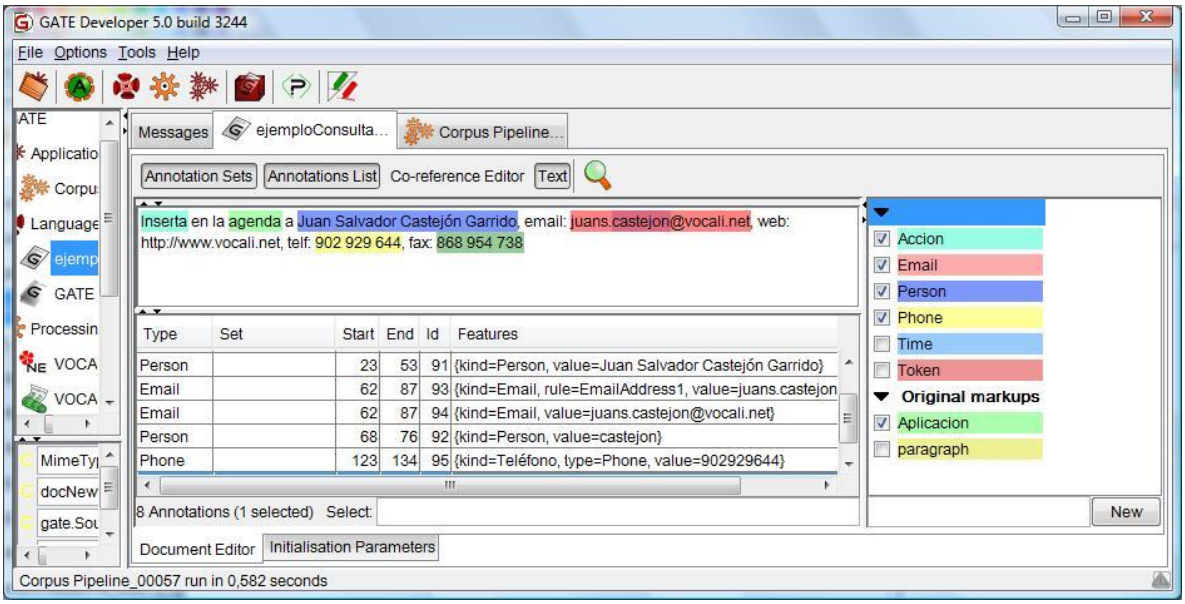


Figura 26. Ejemplo de detección de entidades a partir de los recursos desarrollados.

III.2. SUBSISTEMA ASR

El subsistema ASR es el encargado de procesar la voz del usuario y obtener el texto que será procesado por el módulo NLP con el fin de "comprender" la orden dada por el usuario y ejecutarla.

El reconocimiento de voz todavía es una tecnología imprecisa, por lo que es un campo de investigación en la actualidad con mucha actividad y donde se está avanzando enormemente en los últimos años.

No es el objeto de esta tesis investigar acerca de este tipo de tecnología, sino usarla para conseguir los mejores resultados en el sistema final. Para tal fin, se han tenido que llevar a cabo diferentes acciones que han ido desde la selección del tipo de reconocimiento de voz requerido para esta tesis doctoral a la configuración de los recursos necesarios para obtener los mejores resultados.

III.2.1. TIPO DE RECONOCIMIENTO DE VOZ EMPLEADO

En función del tipo de salida que ofrece el sistema de reconocimiento de voz podríamos clasificar los mismos como sistemas de dictado automático, sistemas de reconocimiento de comandos y sistemas de *word spotting.*

En el caso de los sistemas de dictado automático, el reconocedor de voz tiene la misión de transcribir todo lo que dice el usuario con la mayor precisión posible. Este tipo

de sistemas están pensados para escribir lo que dice el usuario por lo que por general son sistemas que trabajan con grandes vocabularios (decenas de miles de palabras).

Los sistemas de reconocimiento de comandos trabajan con gramáticas predefinidas donde se definen todas las *utterances* que el usuario puede decir. Aunque estas gramáticas pueden ser bastante complejas, en este caso el tamaño del vocabulario que se maneja por lo general suele ser pequeño (varios centenares de palabras). Estos sistemas están pensados para detectar pequeñas frases previamente definidas. Si el usuario dice algo que no se encuentra definido en la gramática, el reconocedor de voz simplemente no lo detectará.

Por último, los sistemas de *word spotting* permiten reconocer palabras previamente definidas en un *stream* de audio. Estos sistemas se usan principalmente en aplicaciones telefónicas de *call center* para detectar palabras clave y llevar al usuario al lugar donde puede encontrar la información que necesita.

En el caso de este trabajo, aunque los comandos son fácilmente identificables, y por tanto definibles, existen comandos que permiten al usuario dictar mensajes completos, o simplemente órdenes como las relacionadas con la agenda que tienen parámetros que pueden contener cualquier valor (como el lugar de una reunión). Por esa razón se ha decidido adoptar una solución mixta donde principalmente se usará un sistema de reconocimiento de voz para dictado pero que contendrá la definición de algunos comandos más sencillos que se puedan representar con gramáticas. De esta forma, si el usuario dice un comando sencillo que encaja con una entrada de alguna gramática, el sistema se decantará por esa opción. Mientras que si el reconocedor de voz no detecta ninguna correspondencia de lo dicho por el usuario con lo definido en una gramática entonces se limitará a transcribirlo. En el caso de los comandos que encajan con la definición en una gramática el propio reconocedor de voz puede hacer de motor NLP ya que al ser la gramática una representación estructurada, se puede devolver el comando ya formado con su significado sin necesidad de motor NLP. Sin embargo, en la mayoría de los casos el sistema se limitará a transcribir la orden del usuario y será el motor NLP el que se encargará obtener su significado, por lo que este subapartado se centrará en la descripción de estos.

III.2.2. EVALUACIÓN DE DIFERENTES SOLUCIONES

Durante el desarrollo de este trabajo el reconocimiento de voz todavía no estaba disponible en los dispositivos móviles como ocurre en la actualidad. Mientras se trabajaba en el mismo no existían sistemas como Apple Siri o Google Now por lo que el reconocimiento de voz tenía que hacerse a través de otro proveedor, por lo que se estudiaron diferentes opciones que se detallan a continuación.

III.2.2.1. SpinVox

SpinVox era un sistema de reconocimiento de voz lanzado para transformar mensajes de voz en SMS. El usuario podía dictar una nota de voz para otra persona y esta última recibiría la transcripción en un mensaje de texto (SMS).

La calidad del servicio era muy buena puesto que el equipo de SpinVox había desarrollado un modelo lingüístico y acústico tras procesar millones de notas de voz que transcribieron transcriptores humanos. Eso les permitió tener un sistema muy precio para el dominio donde se empleaba.

SpinVox era un sistema puramente de transcripción y no permitía definir gramáticas, comandos o enriquecer el vocabulario de ninguna manera.

La compañía fue fundada en Reino Unido en el año 2003 y fue vendida en el año 2009 a la multinacional y gran gigante del sector del reconocimiento de voz: Nuance Communications.

III.2.2.2. Loquendo

Loquendo era un sistema de reconocimiento de voz muy popular en el uso en sistemas telefónicos para *call centers*. Debido a esto, su fuerte estaba en los sistemas orientados a comandos mediante el uso de gramáticas, y en los sistemas de *word spotting*, aunque disponía también de un sistema de transcripción para vocabularios pequeños (< 5000 palabras).

Por estas razones Loquendo no ofrecía un modelo lingüístico por defecto y tenía que definirse por el integrador.

Esta empresa italiana fue adquirida al igual que en el caso de SpinVox por la empresa Nuance Communication.

III.2.2.3. Dragon Naturally Speaking

Se trata del producto estrella de Nuance Communication. Es un sistema de transcripción muy popular y con un rico vocabulario disponible para distintos idiomas. Además, dispone de funcionalidad para enriquecer el vocabulario existente y para entrenar el perfil de voz del usuario y mejorar la precisión, lo que podía ser útil en un momento determinado.

Sin embargo, no es un sistema pensado para funciona como un servicio, sino como una aplicación y hacía que el despliegue de este producto en un entorno de nube para dar servicio fuese demasiado complicado.

III.2.2.4. CMU Sphinx

Las soluciones comentadas anteriormente son comerciales y por tanto tienen un coste económico en el caso de emplearse en un sistema como el desarrollado en esta tesis.

CMU Sphinx es un sistema de reconocimiento de voz muy popular y potente desarrollado por la Universidad de Carnegie Mellon. Este sistema ofrece todos los tipos de funcionamiento: comandos definidos con gramáticas, *word spotting* o dictado, aunque no permite que el sistema pueda funcionar en varios modos simultáneamente.

Sphinx es un sistema muy versátil y dispone de una tecnología en el estado del arte actual. El único problema que tiene es que, a diferencia de los sistemas comerciales, no incluye modelos acústicos ni lingüísticos, de forma que tienen que ser desarrollados por el integrador. Desarrollar un modelo lingüístico es bastante complicado pero es algo al alcance de esta tesis puesto que es relativamente sencillo construir un corpus del español para generar ese modelo, aunque es muy complicado que ese corpus se adapte a las necesidades del dominio planteadas en este trabajo. Por otro lado, desarrollar un modelo acústico es bastante complejo porque se necesitan centenares de audio (y su correspondiente transcripción) para entrenar al sistema. Además, para que ese modelo acústico sea bueno, es necesario que las grabaciones de voz de voz sean de una muestra poblacional diversa (sexo, edad, regiones, acentos,…).

Aunque existen modelos acústicos y lingüísticos gratuitos, al menos para el español están muy lejos de llegar a las necesidades de tasa de acierto necesarios para el correcto funcionamiento del sistema.

III.2.3. SOLUCIÓN DE RECONOCIMIENTO DE VOZ EMPLEADA

Finalmente la solución seleccionada fue SpinVox porque cumplía las necesidades mínimas (conversión voz a texto o transcripción) y por razones comerciales, que se explican a continuación.

SpinVox firmó un contrato con Vodafone en España en el año 2009 para crear un servicio que en España se llamó DictaSMS. Se consideró por tanto que los resultados de este trabajo podían interesar comercialmente a Vodafone para incrementar el uso de este servicio DictaSMS. La idea era la siguiente: cada vez que el usuario cliente de Vodafone empleaba el servicio DictaSMS, Vodafone le cobraba el coste del SMS más otra cantidad (similar al coste de otro SMS) por el uso del servicio de transcripción. Este servicio era poco utilizado y la idea que se le planteó al departamento de I+D de Vodafone fue que la aplicación resultante de este trabajo podría emplear el mismo servicio DictaSMS para capturar la voz del usuario y procesarla, de forma que por cada uso de la aplicación Vodafone tendría un beneficio directo.

III.2.4. CONSIDERACIONES DE LA SOLUCIÓN SELECCIONADA

El servicio DictaSMS ofrecido por Vodafone tenía lo necesario para poder realizar el sistema pero contaba también con importantes limitaciones que se tuiveron que salvar.

La principal limitación consistía en que el servicio DictaSMS era un servicio asíncrono, es decir, no era un sistema en tiempo real y Vodafone no podía garantizar el tiempo que tardaría el servicio en devolver la transcripción. De esta forma el usuario podía decir una orden y podía pasar un tiempo elevado hasta obtener la respuesta.

Las pruebas que se hicieron indicaban que el servicio DictaSMS podía tardar hasta 5 minutos en devolver el resultado de la transcripción. De esta forma, la *app* para *smartphone* desarrollada tenía que poder funcionar en segundo plano y esperar a obtener el SMS enviado por el servicio DictaSMS para procesarlo y ejecutar el comando, como se describe en el apartado III.4. Aplicación móvil.

Otra limitación que tenía SpinVox es que no permitía enriquecer el diccionario. Puesto que era un servicio ofrecido por Vodafone, todo el servicio residía en sus instalaciones y no permitían modificaciones particulares del servicio. Esta limitación no se pudo salvar aunque los resultados obtenidos con el servicio genérico de DictaSMS eran suficientemente buenos para que no supusies una limitación que hiciese inviable el resultado final de la tesis doctoral.

III.3. SUBSISTEMA NLP

III.3.1. DESCRIPCIÓN GENERAL DEL SUBSISTEMA

En este apartado el subsistema capaz de procesar una orden en lenguaje libre para obtener un comando estructurado. Obviamente, el principal componente de este subsistema es el motor de procesamiento del lenguaje natural (NLP) que puede procesar un comando completo y obtener su significado y generar el comando estructurado en XML según el formato que se describe en el correspondiente apartado.

Para el desarrollo del componente principal del sistema que permita procesar los comandos de texto en lenguaje natural se ha utilizado el *framework* GATE y el lenguaje de programación Java. Este motor de se ha obtenido integrando cada uno de los módulos desarrollados como el módulo de detección de expresiones temporales o el módulo de desambiguación.

Además, en este componente se incluyen otros módulos que permiten la construcción del comando estructurado a partir de las anotaciones obtenidas por los distintos módulos.

En el siguiente apartado se describe en detalle la arquitectura del sistema desarrollado.

III.3.1.1. Arquitectura

Este subsistema es el componente principal del sistema puesto que permite procesar los comandos de texto en lenguaje natural dictados por el usuario para la obtención de comandos estructurados que permitan registrar la operación del usuario en el terminal móvil. Para ello se ha empleado las tecnologías descritas en esta tesis y el *framework* GATE para desarrollo de aplicaciones de procesamiento del lenguaje natural. Además, esta solución se basa en la utilización e integración de los recursos lingüísticos desarrollados y los módulos de reconocimiento de expresiones temporales y del sistema de inferencia para la resolución de ambigüedades.

En primer lugar se realizó un estudio de diversas tecnologías de ingeniería lingüística para determinar las herramientas de Procesamiento del Lenguaje Natural (NLP) necesarias para el correcto procesamiento y comprensión semántica de la orden verbal.

En esta evaluación se decidió utilizar la herramienta GATE sobre la cual se desarrolló una aplicación compuesta por distintos módulos como muestra la Figura 27.

La entrada a esta aplicación es un comando escrito en texto en lenguaje natural y la salida es un comando estructurado en el formato XML siguiendo el esquema indicado en el correspondiente apartado de este capítulo, y que después se procesará en la aplicación cliente del móvil.

Como se puede observar en la Figura 27, el sistema se basa en cuatro módulos principales: *Preprocesamiento del texto*, *detección y resolución de expresiones temporales*, *detección y resolución de ambigüedades* y *selección del mejor comando y creación del comando en XML*. A continuación se explican cada uno de estos componentes.

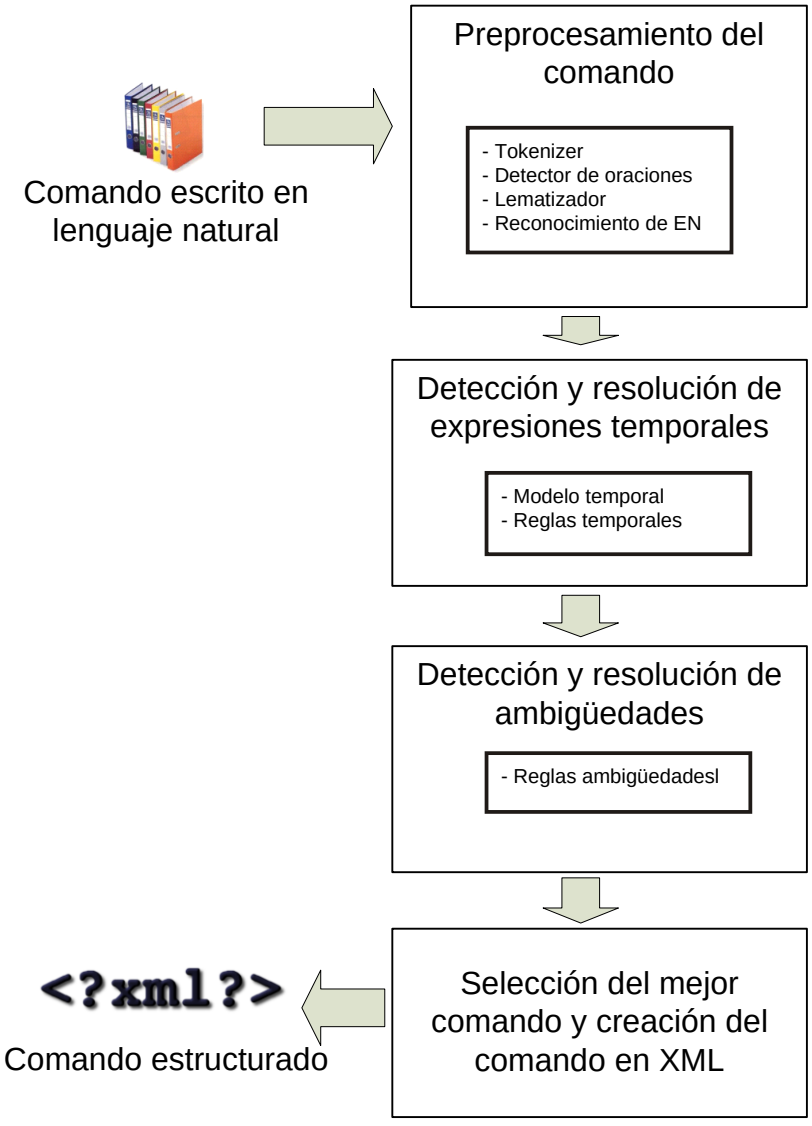


Figura 27. Arquitectura del motor de procesamiento de lenguaje natural.

III.3.1.1.1. Preprocesamiento del texto

Este módulo tiene el objetivo de preprocesar el texto y para ello hace uso de un conjunto de herramientas como un tokenizador, un detector de oraciones, un lematizador y un sistema de reconocimiento de entidades nombradas basado en los recursos lingüísticos desarrollados y explicados más adelante. El sistema de reconocimiento de entidades nombradas se basa en primero un conjunto de listas o *gazetters* y luego un conjuntos de reglas JAPE. Para la extracción de las anotaciones para las listas se ha utilizado un *gazetteer* específico para las necesidades de esta tesis, de igual forma que se ha desarrollado un *NE Transducer* para la ejecución de las reglas JAPE.

Como se ha comentado anteriormente, GATE ha sido la herramienta elegida para poder implementar este módulo de reconocimiento de órdenes en lenguaje natural y para ello se ha creado un *pipeline* en GATE que permite ejecutar todos estos módulos que se han comentado en el párrafo anterior. En la figura se muestra una captura de pantalla del *pipeline* creado en GATE.

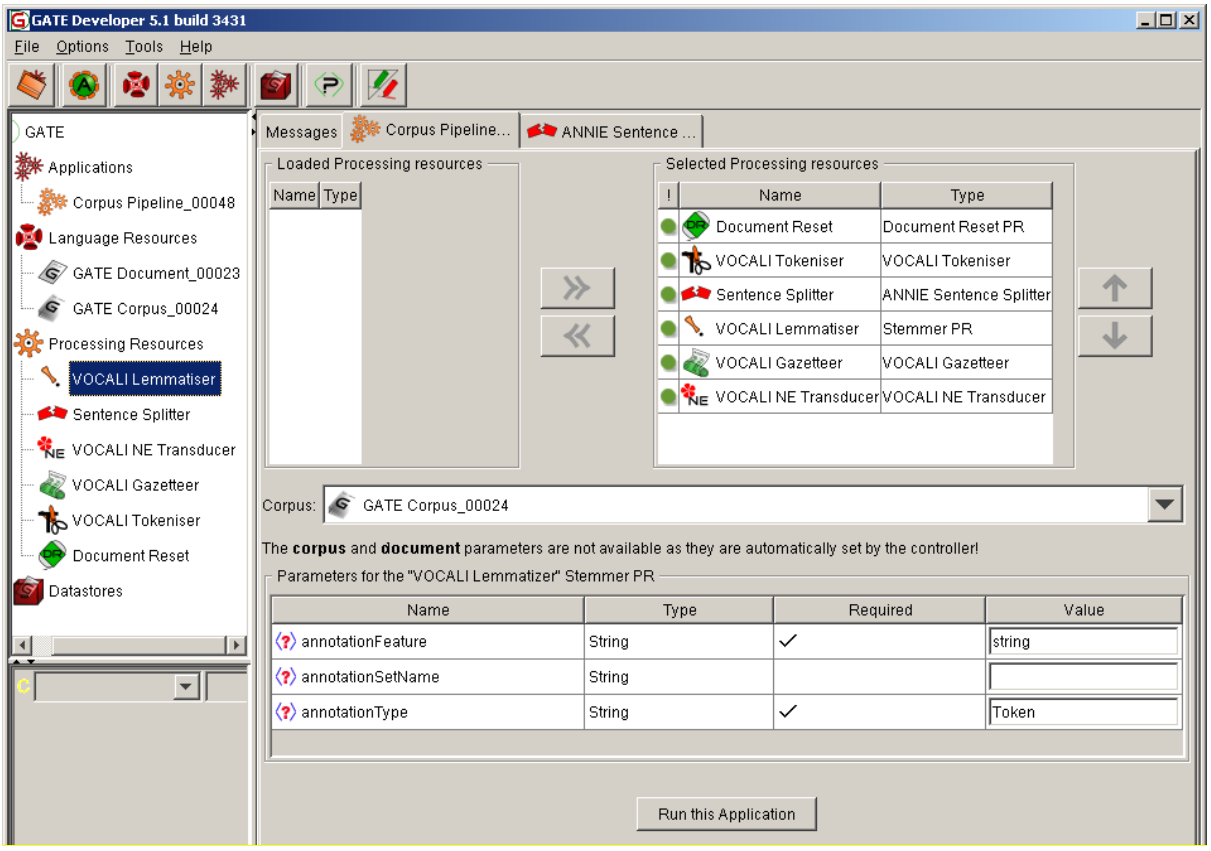


Figura 28. Pipeline de GATE para el procesamiento de órdenes verbales.

Las fases incluidas en el pipeline y que se pueden observar en la Figura 28 son las siguientes:

- Document Reset. Este módulo elimina todas las anotaciones anteriores y deja el pipeline listo para ser usado.
- VOCALI Tokeniser. Este módulo divide el texto en tokens que pueden ser procesados individualmente. Este módulo se describe más en detalle en el documento R2.2-recursos ”inguisticos.
- Sentence Splitter. Este módulo se basa en un módulo ya desarrollado en GATE y permite obtener las distintas oraciones del texto.
- VOCALI Lemmatizer. Este módulo obtiene la forma lematizada de cada token obtenido en el segundo módulo.
- VOCALI Gazetteer. Este módulo identifica los fragmentos de texto que están incluidas en listas o lexicones que se han definido para poder extraer conocimiento de los comandos verbales.
- VOCALI NE Transducer. Este módulo ejecuta las reglas JAPE definidas para la extracción de entidades nombradas y anotaciones para poder identificar el contenido del comando verbal. Este módulo hace uso de los módulos anteriores y estas reglas JAPE se definen con respecto a las anotaciones obtenidas por los módulos anteriores.

III.3.1.1.1.1 Implementación en Java

Debido a las características de la aplicación, el *pipeline* de GATE descrito en la Figura 28, puede cambiar significativamente y con relativa frecuencia durante la implementación del sistema. Por esa razón, se consideró que el motor NLP cargue un *pipeline* de GATE que previamente se haya exportado.

GATE dispone de una aplicación gráfica que nos permite crear *pipelines* y cargar documentos para hacer pruebas desde el entorno, sin necesidad de programar ninguna aplicación que use las librerías de GATE. Por ejemplo, en la Figura 29 se muestra un ejemplo de ejecución de pruebas de entorno dentro del framework GATE.

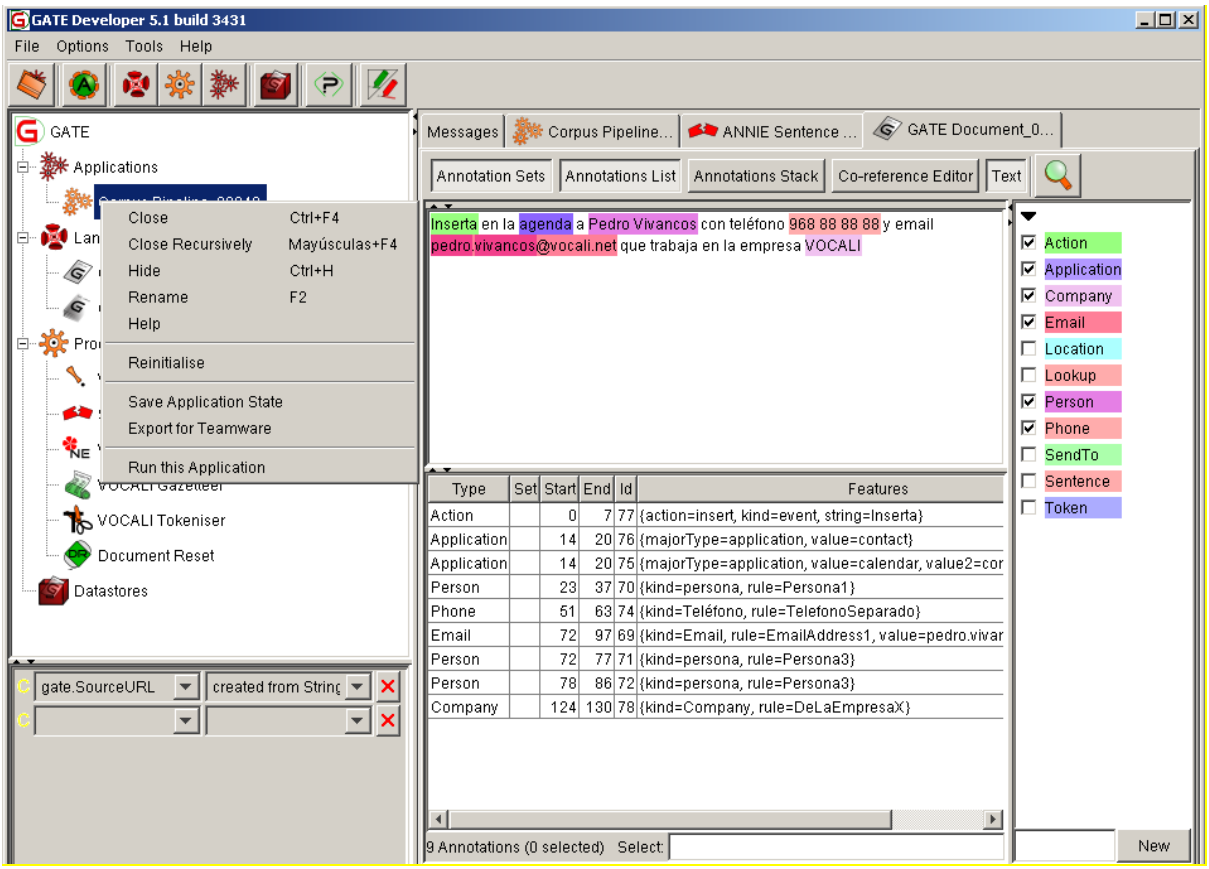


Figura 29. Ejemplo de pruebas con la aplicación gráfica del Framework GATE.

Además, GATE también permite exportar esa aplicación o *pipeline* en un formato XML que luego podemos cargar desde nuestra aplicación Java. De esta manera hacemos el motor NLP más independiente de los distintos procesos que posteriormente se usen en el *pipeline*.

Para exportar el *pipeline* de GATE tendremos que seleccionar dicha aplicación pulsando con el botón derecho del ratón sobre ella y seleccionando la opción "*Save application state*" (ver Figura 29) que generará un fichero con extensión `.xgapp` con información del *pipeline* representada en XML.

Una aplicación GATE guardada en el formato `.xgapp` puede recuperarse con la API de GATE. Existirá por tanto una clase en el programa que se encargará de cargar una aplicación GATE (en definitiva un *pipeline*) con los recursos (*Tokenizer*, *Gazetteer*,...) que se usarán para procesar el texto. Cargar una aplicación de esta manera tiene la ventaja de que hacemos el motor independiente de los recursos utilizados, más teniendo en cuenta que al tratarse de un primer prototipo, los recursos GATE cambiarán con frecuencia durante el desarrollo del mismo y en futuras evoluciones del sistema.

La documentación de cómo cargar una aplicación exportada viene especificada en la documentación de GATE, aunque resumiendo se deben utilizar las siguientes instrucciones:

```
Gate.init();
CorpusController controller = (CorpusController)
PersistenceManager.loadObjectFromFile(new
File("smartmobilevoice.xgapp"))
//La aplicación GATE viene almacenada en el fichero .xgapp que debe
estar dentro del propio proyecto.
```

Figura 30. Código Java de inicialización de GATE.

III.3.1.1.2. Detección y resolución de expresiones temporales

El objetivo fundamental de este módulo es detectar las expresiones temporales tanto de fechas como de horas. La solución planteada se basa en el trabajo presentado en (Saquete Boró, 2005) y en un esquema de anotaciones temporales basado en TimeML. Este componente modela un sistema basado en conocimiento para la detección y resolución de expresiones temporales usando la infraestructura GATE y en particular usando reglas JAPE que son apropiadas para detectar, y posteriormente anotar, expresiones temporales escritas como “el lunes de la semana que viene”, “a las 6 y media de la tarde”, “entre las 10 y las 12 de la noche”, etc.

Además, se ha desarrollado un modelo temporal, basado en reglas lógicas capaces de resolver complejas y ambiguas expresiones temporales (p.e. “el primer lunes del mes que viene” se refiere al 01/11/2011). Para ello se ha desarrollado un módulo software en JAVA que permite detectar y anotar todo tipo de expresiones temporales.

III.3.1.1.3. Detección y resolución de ambigüedades

El objetivo principal de este módulo es resolver las posibles ambigüedades derivadas de la naturaleza del lenguaje natural que puedan confundir al sistema en la detección del comando a realizar. Por ejemplo, el comando “recuérdame que tengo que enviar un correo electrónico a Antonio diciéndole que ya está reservado el hotel para las vacaciones” está asociado a la creación de una nueva tarea en la aplicación del móvil, no al envío de un correo electrónico.

Además, es muy probable que una misma palabra pueda representar varios significados. En nuestro caso la palabra “correo” puede significar o bien la aplicación del correo electrónico o bien hacer referencia a los datos de un correo de una persona.

Otras palabras como “Murcia” pueden ser a su vez una localización o bien un apellido de una persona.

La solución planteada se basa se basa en el trabajo presentado en (Vázquez Pérez, 2009) y modela un sistema basado en conocimiento, reglas y heurísticas para la desambiguación usando la infraestructura GATE y en particular usando reglas JAPE que son apropiadas para detectar, y posteriormente anotar, las expresiones ambiguas y en base al contexto poder decidir qué sentido será el correcto en el comando del dispositivo móvil.

Para ello, se ha desarrollado un modelo de desambiguación, basado en heurísticas para poder apoyar las reglas JAPE definidas. Dicho módulo se ha desarrollado un módulo software en JAVA que permite desambiguar dentro del dominio de aplicación de este trabajo.

La descripción detallada de este módulo se puede consultar en el apartado Resolución de ambigüedades.

III.3.1.1.4. Selección del mejor comando y creación del comando en XML.

Este módulo es el encargado de procesar las anotaciones no ambiguas obtenidas del módulo anterior y transformar dichas anotaciones en un comando estructurado XML según el XML Schema descrito en el apartado XML Schema empleado.

Este módulo primero debe comprobar si a partir de las anotaciones obtenidas se puede crear un comando. Para desarrollar esta comprobación se ha desarrollado un conjunto de heurísticas que indican qué elementos son necesarios para poder formar dicho comando. Por ejemplo, un comando para escribir un email deberá tener identificada una anotación *Action* "Crear", una anotación *Application* "email" y una anotación que contenga el destinatario del correo electrónico *SendTo*. Otro ejemplo sería que para poder insertar un contacto en la agenda se debe haber identificado como *Application* “agenda” la acción (anotación *Action*) debería ser “insertar” y al menos debería aparecer una *Persona*.

El generador XML se ha implementado utilizando JAXB. Esta librería permite, a partir de un esquema XML generar un conjunto de clases Java que se corresponden con el mismo. Además, automatiza la serialización de estos objetos de acuerdo al propio esquema. Se ha utilizado JAXB para generar las clases a partir del esquema (se deberá

hacer en tiempo de compilación). Estas clases son las que el motor NLP devolverá como salida de su procesamiento y que el generador XML transformará a XML mediante JAXB.

En el Anexo I. Ejemplos de comandos se muestran algunos ejemplos de comandos XML que deberían obtenerse a partir de órdenes escritas en lenguaje natural. Estos ejemplos se han dividido por las aplicaciones *calendario*, *agenda*, *email*, *SMS*, *tareas* y *notas*.

III.3.2. RECURSOS LINGÜÍSTICOS EMPLEADOS

En este apartado se detallan los recursos lingüísticos desarrollados para la detección de entidades relevantes en los comandos verbales dentro del subsistema motor NLP para el dominio de aplicación, que como se ha comentado a lo largo de esta tesis, es el control de los dispositivos móviles mediante lenguaje natural.

El framework utilizado para la detección de entidades nombradas y otras cosas relevantes del dominio ha sido GATE (General Architecture for Text Engineering, http://gate.ac.uk/).

A continuación se describe el framework por encima centrándonos en los apartados relacionados con la construcción de los recursos lingüísticos.

III.3.2.1. Estructura de los recursos lingüísticos

Para el correcto desarrollo de los recursos lingüísticos se ha creado un nuevo CREOLE que contiene los siguientes módulos de procesamiento de GATE:

- VOCALI Tokenizer
- VOCALI Gazetter
- VOCALI NE Transducer

En la Figura 31 se puede ver una captura de pantalla de GATE procesando un comando de voz e identificando los tipos distintos de entidades en dicho comando.

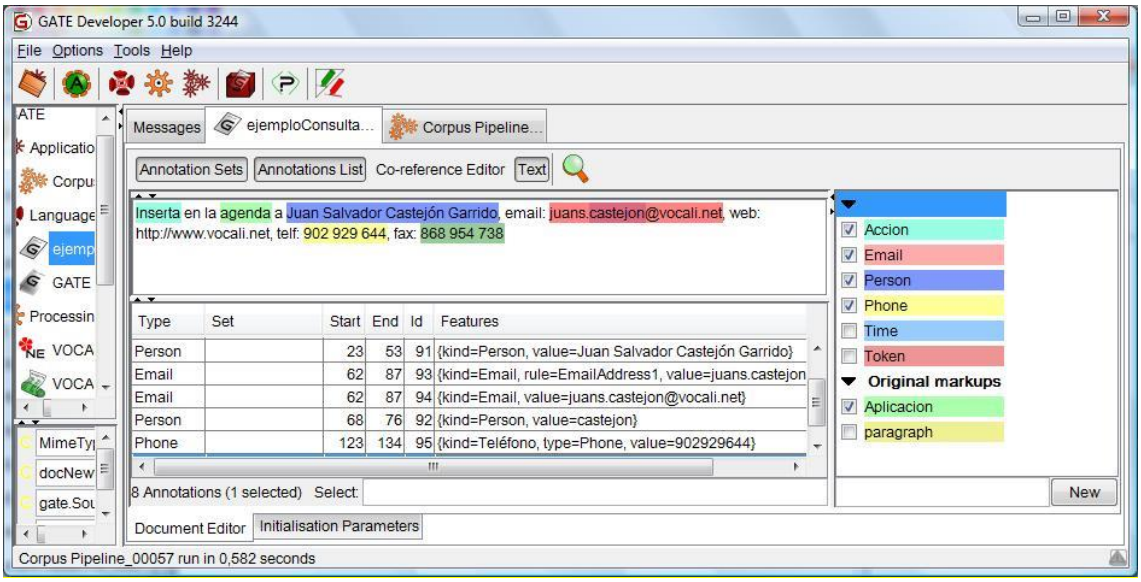


Figura 31. Ejemplo de detección de entidades a partir de los recursos desarrollados

Una descripción parcial del fichero que define el CREOLE puede verse en la siguiente figura.

```
<RESOURCE>
	<NAME>VOCALI Gazetteer</NAME>
	<CLASS>gate.creole.gazetteer.DefaultGazetteer</CLASS>
	<PARAMETER NAME="document" RUNTIME="true"
	COMMENT="The document to be processed">
		gate.Document
	</PARAMETER>
<PARAMETER NAME="gazetteerFeatureSeparator" DEFAULT="&amp;" COMMENT="The
character used to separate features for entries in gazetteer lists. Accepts
strings like &quot;\t&quot; and will unescape it to the relevant character. If
not specified, this gazetteer does not support extra features."
OPTIONAL="true">
java.lang.String
</PARAMETER>
<PARAMETER NAME="listsURL" DEFAULT="gazetteer/lists.def" COMMENT="The URL to
the file with list of lists" SUFFIXES="def">
		java.net.URL
	</PARAMETER>
<PARAMETER NAME="caseSensitive" DEFAULT="false" COMMENT="Should this gazetteer
diferentiate on case">
java.lang.Boolean
</PARAMETER>
<PARAMETER NAME="encoding" DEFAULT="UTF-8" COMMENT="The encoding used for
reading the definitions">
		java.lang.String
	</PARAMETER>
```

```
<PARAMETER NAME="annotationSetName" RUNTIME="true" COMMENT="The annotation set
to be used for the generated annotations" OPTIONAL="true">
java.lang.String
</PARAMETER>
<PARAMETER NAME="wholeWordsOnly" DEFAULT="true" COMMENT="Should this gazetteer
only match whole words" RUNTIME="true">
            java.lang.Boolean
      </PARAMETER>
      <ICON>gazetteer</ICON>
</RESOURCE>
<RESOURCE>
      <NAME>VOCALI NE Transducer</NAME>
      <CLASS>gate.creole.ANNIETransducer</CLASS>
      <PARAMETER NAME="document" RUNTIME="true" COMMENT="The document to be
processed">
            gate.Document
      </PARAMETER>
<PARAMETER NAME="inputASName" RUNTIME="true" COMMENT="The annotation set to be
used as input for the transducer" OPTIONAL="true">
            java.lang.String
      </PARAMETER>
<PARAMETER NAME="outputASName" RUNTIME="true" COMMENT="The annotation set to
be used as output for the transducer" OPTIONAL="true">
            java.lang.String
      </PARAMETER>
<PARAMETER NAME="grammarURL" COMMENT="The URL to the grammar file"
DEFAULT="NE/main.jape" SUFFIXES="jape">
            java.net.URL
      </PARAMETER>
<PARAMETER NAME="encoding" DEFAULT="UTF-8" COMMENT="The encoding used for
reading the grammar">
            java.lang.String
      </PARAMETER>
      <ICON>ne-transducer</ICON>
</RESOURCE>
```

Figura 32. Parte del fichero CREOLE con el plugin de GATE desarrollado.

En los siguientes subapartados se describen cada uno de estos módulos.

III.3.2.1.1. VÓCALI Tokenizer

El tokenizer que proporciona GATE por defecto está centrado en el idioma inglés y dispone de varios bugs, por esa razón para esta tesis doctoral se ha desarrollado un nuevo tokeniser para español que soluciona dichos bugs. Para ello, se ha definido un conjunto de reglas que implementan el tokeniser y que definen como se delimitan las

palabras y otros símbolos en el idioma español. En la Figura 33 se muestra un extracto de este fichero donde puede verse cómo se definen las palabras, números, espacios en blanco, símbolos, etc.

III.3.2.1.2. VÓCALI Gazetteer

Como se ha comentado en la sección 1, los gazetters representan lexicones o listas de términos de un determinado dominio. A continuación se va a explicar la estructura de los lexicones desarrollados en este trabajo de investigación.

III.3.2.1.3. Tipo de listas Gazetter

Se han desarrollado dos tipos distintos de lexicones del Gazetteer: las que son generales e independientes del trabahi y las que son válidas sólo para este trabajo de investigación. Para ello se insertan en el directorio /etc/gate/gazetter las listas que son generales y en /etc/gate/gazetterSmartMobileVoice las que son de interés para la tesis doctoral, por lo que en cada directorio habrá definido un fichero lists.def.

Todos los ficheros de listas deben tener nombres representativos que identifiquen claramente la información que contiene. La estructura de nombres que se utilizará será la siguiente:

[prefijo]_[subprefijo]_[nombre_lista].lst

El prefijo deberá identificar claramente el ámbito para el que está definida la lista. Por ejemplo actualmente se tienen los siguientes prefijos:

- addr_ : Recursos para identificar "address"
- company_ : Recursos para identificar "company"
- loc_ : Recursos para identificar "location"
- number_ : Recursos para identificar "number"
- person_ : Recursos para identificar "person"
- stopword_ : Recursos para identificar "stopword"
- time_ : Recursos para identificar expresiones temporales "date" o "time"

```
#words#
// a word can be any combination of letters, including hyphens,
// but excluding symbols and punctuation, e.g. apostrophes
// Note that there is an alternative version of the tokeniser that
// treats hyphens as separate tokens
```

```
"UPPERCASE_LETTER" (LOWERCASE_LETTER (LOWERCASE_LETTER|FORMAT)*)* >
Token;orth=upperInitial;kind=word;
"UPPERCASE_LETTER" (DASH_PUNCTUATION|FORMAT)* (UPPERCASE_LETTER|FORMAT)+ >
Token;orth=allCaps;kind=word;
"LOWERCASE_LETTER" (LOWERCASE_LETTER|FORMAT)* >
Token;orth=lowercase;kind=word;

// MixedCaps is any mixture of caps and small letters that doesn't
// fit in the preceding categories

("LOWERCASE_LETTER" "LOWERCASE_LETTER"+"UPPERCASE_LETTER"+ \
 (UPPERCASE_LETTER|LOWERCASE_LETTER)*)|\
("LOWERCASE_LETTER" "LOWERCASE_LETTER"*"UPPERCASE_LETTER"+\
 (UPPERCASE_LETTER|LOWERCASE_LETTER|DASH_PUNCTUATION|FORMAT)*)|\
("UPPERCASE_LETTER" (DASH_PUNCTUATION)* "UPPERCASE_LETTER"
(UPPERCASE_LETTER|LOWERCASE_LETTER|DASH_PUNCTUATION|FORMAT)*\
 ("LOWERCASE_LETTER")+
(UPPERCASE_LETTER|LOWERCASE_LETTER|DASH_PUNCTUATION|FORMAT)*)|\
("UPPERCASE_LETTER" "LOWERCASE_LETTER"+ ("UPPERCASE_LETTER"+
"LOWERCASE_LETTER"+)+)|\
 ((UPPERCASE_LETTER)+ (LOWERCASE_LETTER)+ (UPPERCASE_LETTER)+)\
> Token;orth=mixedCaps;kind=word;

(OTHER_LETTER|COMBINING_SPACING_MARK|NON_SPACING_MARK)+ >Token;kind=word;type=
other;

#numbers#
// a number is any combination of digits
"DECIMAL_DIGIT_NUMBER"+ >Token;kind=number;

#whitespace#
(SPACE_SEPARATOR) >SpaceToken;kind=space;
(CONTROL) >NewLineToken;kind=control;

#symbols#
(MODIFIER_SYMBOL|MATH_SYMBOL|OTHER_SYMBOL) > Token;kind=symbol;
CURRENCY_SYMBOL > Token;kind=symbol;symbolkind=currency;

#punctuation#
(DASH_PUNCTUATION|FORMAT)(DASH_PUNCTUATION|FORMAT)+ >Token;kind=punctuation;su
bkind=2dashpunct;
(DASH_PUNCTUATION|FORMAT) >Token;kind=punctuation;subkind=dashpunct;

#punctuaction - if there are more than one puntuaction system they have to be
treated as just one token
#(CONNECTOR_PUNCTUATION|OTHER_PUNCTUATION)>Token;kind=punctuation;
```

```
"START_PUNCTUATION" >Token;kind=punctuation;position=startpunct;
"END_PUNCTUATION" >Token;kind=punctuation;position=endpunct;
```

Figura 33. Parte del fichero tokenizer.rules que describe el VOCALI Tokenizer desarrollado.

Además, las listas que comiencen por el prefijo "mobilevoice" serán dependientes del trabajo aplicación y podrán contener otro prefijo "context":

- mobilevoice_ : Recursos relacionados con la aplicación.
- mobilevoice_context_ : Recursos que ayuden a determinar expresiones que ayudan a determinar que viene a continuación (p.e. "añade al contacto Pepe García con teléfono 555555555" => la palabra "teléfono" debería estar en una lista de contexto pues nos indica que lo que viene a continuación es un teléfono, aunque no case exactamente con un patrón de los que tenemos).

III.3.2.1.3.1 Tipos de majorType y minorType

El majorType y minorType de las listas debe ser representativo y evitar al máximo ambigüedad y posibles confusiones. El majorType debe representar la entidad general a la que representa y el minorType especificar. Actualmente tenemos los siguientes majorType:

- Tipos para lexicones generales
    - address - Para representar calles y direcciones
    - company - Para representar empresas y organizaciones
    - location - Para representar localizaciones
    - number - Para representar números
    - person_first - Para identificar nombres de persona
    - person_familymembers - Para identificar el parentesco y los miembros de la familia (hermanos, padres, etc)
    - person_ending - Para identificar apellidos
    - person_abbr - Para identificar abreviaturas de títulos de nombres de persona (Sr., etc)
    - stopword - Para identificar stopwords
    - timex - Para identificar expresiones temporales
- Tipos para lexicones relacionados directamente con la tesis doctoral.
    - action - Para identificar el tipo de acción
    - application- Para identificar el tipo de aplicación

- context_application - Para identificar palabras en el contexto que ayuden a determinar la aplicación como reunión, cita, etc.
- context - Recursos que ayuden a determinar expresiones que ayudan a determinar que viene a continuación (p.e. "añade al contacto Pepe García con teléfono 555555555" => la palabra "teléfono" debería estar en una lista de contexto pues nos indica que lo que viene a continuación es un teléfono, aunque no case exactamente con un patrón de los que tenemos).
  - context:body - Para identificar el cuerpo del mensaje
  - context:email - Para identificar emails
  - context:empty_subject - Para identificar asuntos vacíos del mensaje
  - context:phone - Para identificar teléfonos
  - context:subject - Para identificar el asunto del mensaje

III.3.2.1.3.2 List.def

Un extracto del fichero list.def actual se puede ver en la Figura 34. Como se puede ver se han creado más de 50 lexicones distintos que incluyen recursos lingüísticos de nombres de persona, ciudades, empresas y organizaciones, etc.

```
spanish_male_firstname.lst:person_first:male
spanish_male_firstname_diminutive.lst:person_first:male
spanish_female_firstname.lst:person_first:female
spanish_female_firstname_diminutive.lst:person_first:female
spanish_surname.lst:person_ending
catalan_female_firstname.lst:person_first:female
catalan_female_firstname_diminutive.lst:person_first:female
catalan_male_firstname.lst:person_first:male
catalan_male_firstname_diminutive.lst:person_first:male
spanish_country.lst:location:country
spanish_country_capital.lst:location:city
abreviaturas.lst:abreviaturas
dayoftheweek.lst:time:dayoftheweek
months_lower.lst:time:month
numbers.lst:number
ordinal.lst:ordinal
street.lst:tipos_calle
phone_prefix.lst:números_teléfono
```

```
mountain.lst:location:montañas
regiones_europa.lst:location:regiones_europa
tvcompany.lst:company:cadenas_tv
zonas_geopolíticas.lst:location:zonas_geopolíticas
spanish_city.lst:location:ciudades_de_España
estaciones.list:time:estaciones_del_año
water.mar_oceano.lst:location:mares_océanos
water.golfo_bahía.lst:location:golfos_bahías
loc_relig.lst:location:lugares_religiosos
timesofday.lst:timex:timesofday
timesofday_pre.lst:timex:timesofday_pre
timesofday_post.lst:timex:timesofday_post
vocaliminutes.lst:timex:minutes
date.lst:time:mediciones_tiempo
organization.lst:company
loc_generalkey.lst:location
emisoras_radio.lst:company:emisoras_esp_radio
person_ppo.lst:abreviatura_nombre
loc_murcia.municipios.lst:location:municipios_región_de_Murcia
loc_murcia.pedanias.lst:location:pedanias_murcia
loc_murcia.pedan-
dip.lst:location:pedanias_diputaciones_región_de_Murcia
loc_murcia.comarcas.lst:location:comarcas_región_de_Murcia
fechasfuzzy.lst:time:fechafuzzy
abreviaturasdireccion.lst:abrdireccion
event.lst:event
company.lst:company
application.lst:application
parentesco.lst:person_familymembers
determiner.lst:stopword:determiner
preposition.lst:stopword:preposition
conjunction.lst:stopword:conjunction
```

Figura 34. Parte del fichero lists.def desarrollado.

III.3.2.1.4. VOCALI NE Transducer

Como se ha comentado anteriormente, los recursos generados en el Transducer mediante expresiones regulares implementadas en reglas JAPE, desarrollan los métodos para la extracción de entidades nombradas y anotaciones en un texto. A continuación se va a explicar la estructura de las reglas desarrolladas en este trabajo de investigación. Para ello, al igual que con los Gazetteers se han definido dos tipo de ficheros JAPE: los

de entidades nombradas generales y los de entidades nombradas específicos de la tesis doctoral. Para ello se define un fichero "main.jape" en el cual se define cada uno de los recursos que se van ejecutando en orden. En la figura siguiente se puede ver parte de dicho fichero con un conjunto de fases generales y específicas.

Todos los ficheros de reglas JAPE deben tener nombres representativos que identifiquen claramente las anotaciones que van a extraer. La estructura de nombres que se utilizará será la siguiente:

[prefijo]_[nombre_regla].jape

El prefijo deberá identificar claramente el ámbito para el que está definido el recurso. Por ejemplo actualmente se tienen los siguientes prefijos:

- general_ : Recursos para identificar anotaciones generales
- mobilevoice_ : Recursos para identificar anotaciones propias del trabajo como comandos verbales, aplicaciones, etc.

```
MultiPhase: TestTheGrammars
Phases:
mobilevoice_lematizador
general_email
general_email2
general_persona
mobilevoice_guest
general_fecha
general_direccion
general_localizacion
general_telefono
general_intervalofechas
mobilevoice_application
mobilevoice_action
general_company
general_timesofday
general_timesofdayinterval
mobilevoice_contexto
mobilevoice_sendto
mobilevoice_asunto
mobilevoice_mensaje
general_final
general_clean
```

Figura 35. Parte del fichero main.jape desarrollado.

En la Figura 35 se pueden distinguir estos dos tipos de recursos que se explican brevemente a continuación.

- Generales
    - general_email: Identifica direcciones de correo electrónico.
    - general_email2: Identifica direcciones de correo electrónico.
    - general_persona: Identifica personas a partir de las listas definidas en el apartado anterior.
    - general_fecha: Identifica distintos tipos de fechas y expresiones temporales.
    - general_direccion: Identifica direcciones.
    - general_localizacion: Identifica localizaciones como ciudades y países
    - general_telefono: Identifica número de teléfono.
    - general_intervalofechas: Identifica distintos tipos de intervalos entre fechas y expresiones temporales.
    - general_company: Identifica empresas y organizaciones a partir de los lexicones definidos.
    - general_timesofday. Identifica distintos tipos de expresiones temporales-
    - general_timesofdayinterval: Identifica distintos tipos de intervalos entre fechas y expresiones temporales.
    - general_final: Recopila las anotaciones y elimina duplicados.
    - general_clean: Elimina las anotaciones temporales creadas para la identificación de otras anotaciones.
- Específicos de la tesis doctoral
    - mobilevoice_lematizador: Representa un lematizador que permite obtener la forma raíz de las palabras más importantes utilizadas en el ámbito del trabajo de investigación.
    - mobilevoice_guest: Identifica los invitados a una reunión.
    - mobilevoice_application: Identifica la aplicación del móvil en la que se tiene que ejecutar el comando.
    - mobilevoice_action: Identifica la acción que representa el comando.

- mobilevoice_contexto: Identifica información de contexto para poder desambigüar y obtener otra información.
- mobilevoice_sendto: Identifica el destinatario de un comando para escribir un mensaje de correo electrónico o un SMS.
- mobilevoice_asunto: Identifica el asunto en un comando para escribir un correo electrónico.
- mobilevoice_mensaje: Identifica el cuerpo del mensaje dentro de un comando para escribir un correo electrónico o un SMS

A continuación se explican en detalle los ficheros de reglas general_persona.jape y general_telefono.jape

III.3.2.1.4.1 Reglas para obtención de personas (general_persona)

En la Figura 36 se puede observar el contenido de las reglas para detectar personas. Como se puede observar, hay tres reglas para obtener personas utilizando las anotaciones Lookup obtenidas del Gazetteer y las anotaciones Token, NewLineToken y DEFAULT_TOKEN obtenidas del Tokeniser.

La primera regla identifica como entidad Persona, todo el texto formado por al menos un parentesco (opcional), un nombre propio de persona (person_first) y luego de 1 a 3 palabras que sean o bien nombres propios de persona (person_first), apellidos (person_ending) o cualquier palabra que comience con mayúscula.

La segunda regla identifica como entidad Persona, todo el texto formado por al menos un parentesco (opcional), un apellido (person_ending) y luego de 1 a 3 palabras que sean o bien nombres propios de persona (person_first) o apellidos (person_ending).

Por último, la tercera regla identifica como entidad Persona, todo el texto formado por al menos un parentesco (opcional), y luego de 1 a 3 palabras que sean o bien nombres propios de persona (person_first) o apellidos (person_ending).

```
Phase: general_persona
Input:  Token Lookup NewLineToken DEFAULT_TOKEN
Options: control = appelt

Macro: PARENTESCO
(
      {Lookup.majorType == person_familymembers}
```

```
       ({Token.string == "de"})?
)
// Persona rules

Rule:Persona1
Priority: 50
(
  (PARENTESCO)?
  ({Lookup.majorType == abreviatura_nombre})?
 ({Lookup.majorType ==
person_first, !DEFAULT_TOKEN, !NewLineToken, !Token.orth == lowercase})
 (({Lookup.majorType ==
person_first, !DEFAULT_TOKEN, !NewLineToken, !Token.orth == lowercase}|
  {Lookup.majorType ==
person_ending, !DEFAULT_TOKEN, !NewLineToken, !Token.orth == lowercase} |
  {Token.orth == upperInitial}| {Token.orth == allCaps}
   ))[1,3]
 )
:person -->
  :person.Person= {kind = "persona", rule = "Persona1"}

Rule:Persona2
Priority: 30
(
  (PARENTESCO)?
 ({Lookup.majorType ==
person_ending, !DEFAULT_TOKEN, !NewLineToken, !Token.orth == lowercase})
 (({Lookup.majorType ==
person_first, !DEFAULT_TOKEN, !NewLineToken, !Token.orth == lowercase}|
  {Lookup.majorType ==
person_ending, !DEFAULT_TOKEN, !NewLineToken, !Token.orth == lowercase}
   ))[1,3]
 )
:person -->
  :person.Person= {kind = "persona", rule = "Persona2"}

Rule:Persona3
Priority: 10
(
  (PARENTESCO)?
 (({Lookup.majorType == person_first, !DEFAULT_TOKEN, !NewLineToken}|
  {Lookup.majorType == person_ending, !DEFAULT_TOKEN, !NewLineToken}
  ))[1,3]
 )
:person -->
```

```
  :person.Person= {kind = "persona", rule = "Persona3"}MultiPhase:
       TestTheGrammars
```

Figura 36. Parte del fichero general_persona.jape desarrollado.

III.3.2.1.4.2 Reglas para la obtención de teléfonos (general_telefono)

En la Figura 37 se puede observar el contenido de las reglas para detectar números de teléfonos. Como se puede observar, hay cuatro reglas para obtener teléfonos utilizando las anotaciones aciones Token y NewLineToken obtenidas del Tokeniser.

La primera regla identifica como entidad Teléfono, todo el texto formado por un token numérico de longitud 9. Esto pretende identificar los números de teléfonos españoles sin ninguna separación entre los números.

La segunda regla identifica como entidad Teléfono, todo el texto formado por el carater '+' y un token numérico de longitud 10 u 11. Esta regla pretende identificar los teléfonos con prefijo internacional.

La tercera regla identifica como entidad Teléfono, todas las combinaciones de teléfono separado que se pueden dar, es decir, por ejemplo primero un número de 3 dígitos, seguido de otro número de 3 dígitos.

Por último, la regla cuatro identifica como entidad Teléfono, todas las combinaciones de teléfono separado que se pueden dar incluyendo también un prefijo internacional.

```
Phase: general_telefono
Input:  Token NewLineToken
Options: control = appelt


// Teléfono

Rule:TelefonoJunto
Priority: 50
(
 {Token.kind == number , Token.length == 9}
)
:phone -->
  :phone.Phone= {kind = "Teléfono", rule = "TelefonoJunto"}

Rule:TelefonoJuntoConPrefijo
Priority: 50
(
 ({Token.string == "+"})
```

```
 ({Token.kind == number , Token.length == 11}| {Token.kind == number ,
Token.length == 10})
)
:phone -->
  :phone.Phone= {kind = "Teléfono", rule = "TelefonoJuntoConPrefijo"}

Rule:TelefonoSeparado
Priority: 50
(
 {Token.kind == number , Token.length == 3}
(( {Token.kind == number , Token.length == 3,!NewLineToken}
 {Token.kind == number , Token.length == 3,!NewLineToken})|
 ({Token.kind == number , Token.length == 6,!NewLineToken})|
( {Token.kind == number , Token.length == 2,!NewLineToken}
{Token.kind == number , Token.length == 2,!NewLineToken}
{Token.kind == number , Token.length == 2,!NewLineToken}
))
)
:phone -->
  :phone.Phone= {kind = "Teléfono", rule = "TelefonoSeparado"}

Rule:TelefonoSeparadoConPrefijo
Priority: 50
(
 ({Token.string == "+"})
 ({Token.kind == number , Token.length == 1}| {Token.kind == number ,
Token.length == 2})
 {Token.kind == number , Token.length == 3}
(( {Token.kind == number , Token.length == 3,!NewLineToken}
 {Token.kind == number , Token.length == 3,!NewLineToken})|
 ({Token.kind == number , Token.length == 6,!NewLineToken})|
( {Token.kind == number , Token.length == 2,!NewLineToken}
{Token.kind == number , Token.length == 2,!NewLineToken}
{Token.kind == number , Token.length == 2,!NewLineToken}
))
)
:phone -->
  :phone.Phone= {kind = "Teléfono", rule = "TelefonoSeparadoConPrefijo"}
```

Figura 37. Parte del fichero general_telefono.jape desarrollado.

III.3.3. INTERPRETACIÓN DE EXPRESIONES TEMPORALES

En este apartado se detalla el desarrollo del motor de reconocimiento e interpretación de expresiones temporales. El objetivo fundamental de este módulo es detectar las expresiones temporales tanto de fechas como de horas. Para ello se

evaluaron diferentes trabajos y estándares para la anotación y representación de expresiones que se resumen a continuación.

III.3.3.1. Arquitectura de la solución

La solución planteada se basa en el trabajo presentado en (Saquete Boró, 2005) y modela un sistema basado en conocimiento para la detección y resolución de expresiones temporales usando la infraestructura GATE y en particular usando reglas JAPE que son apropiadas para detectar, y posteriormente anotar, expresiones temporales escritas como “el lunes de la semana que viene”, “a las 6 y media de la tarde”, “entre las 10 y las 12 de la noche”, etc.

Además, se ha desarrollado un modelo temporal, basado en reglas lógicas capaces de resolver complejas y ambiguas expresiones temporales (p.e. “el primer lunes del mes que viene” se refiere al 01/11/2011). Para ello se ha desarrollado un módulo software en JAVA que permite detectar y anotar todo tipo de expresiones temporales.

La arquitectura funcional del módulo de reconocimiento e interpretación de expresiones temporales se puede ver en la Figura 38.

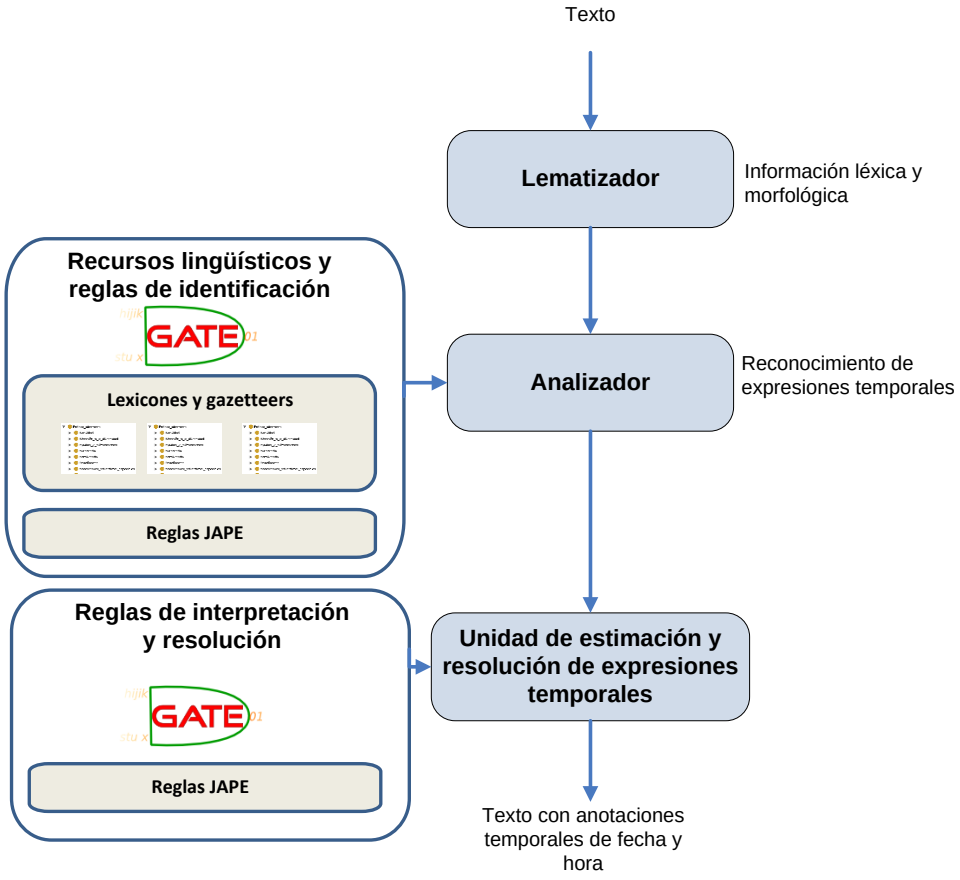


Figura 38. Arquitectura del sistema de detección y resolución de expresiones temporales.

Según la Figura 38 el sistema funciona de la siguiente manera: Primero se realiza un análisis morfológico del texto introducido mediante un lematizador para obtener las raíces de las palabras, seguidamente se ejecuta el analizador que detectará las expresiones temporales del texto utilizando para ello las reglas de identificación de expresiones temporales implementadas en JAPE. Seguidamente el módulo de estimación y resolución de expresiones temporales traducirá la expresión temporal ambigua y subjetiva a una fecha real representada por día, mes y año. A continuación se explica cada uno de estos módulos más detalladamente.

Dentro de la detección de expresiones temporales hay reglas muy sencillas como detectar una fecha escrita en números (por ejemplo 10/10/2010) o reglas complejas para detectar distintas expresiones temporales en lenguaje natural (por ejemplo, el lunes de la semana que viene). Este módulo intenta implementar casi todas las expresiones temporales en castellano del Anexo I que están extraídas del trabajo presentado en (Saquete Boró, 2005)

III.3.3.1.1. Lematizador

Este módulo trata de extraer la información léxica y morfológica del texto introducido. Más concretamente implementa un lematizador que permite obtener el lema a partir de cualquier forma de la palabra que es la raíz de la palabra. Por ejemplo en el caso de las conjugaciones sería el infinitivo y de los adjetivos la forma masculino singular. Este módulo es muy útil para poder contemplar todas las órdenes que contengan verbos en distintos tiempos verbales, y nombres y adjetivos con distintos género y número.

III.3.3.1.2. Analizador

El objetivo de este módulo es la identificación de las expresiones temporales en el texto. Como se ha comentado anteriormente, este módulo se ha implementado haciendo uso de llas listas Gazetteer (lexicones) y las reglas JAPE del NE Transducer de GATE.

A continuación se va a explicar la estructura de los lexicones y reglas desarrolladas para este módulo.

III.3.3.1.2.1 Listas Gazetter

Se han desarrollado dos tipos distintos de lexicones del Gazetteer relacionados con las expresiones temporales. Más concretamente los lexicones relacionados con estas expresiones temporales son los siguientes:

- *timex_dayoftheweek.lst*. Esta lista representa términos que representan a los días de la semana (lunes, martes, miércoles, jueves, etc).
- *timex_months_lower.lst*. Esta lista representa los términos que representan a los meses del año (enero, febrero, marzo, etc.)
- *numbers.lst:number*. Esta lista es general y representa los números escritos en letra que se utilizan para detectar los días del mes.
- *timex_timesofday.lst*. Contiene expresiones temporales para determinar la franja horaria como (esta mañana, esta tarde, al mediodía, etc).
- *timex_timesofday_pre.lst*. Contiene expresiones temporales para determinar la franja horaria como (por la mañana, por la tarde, etc).
- *timex_timesofday_post.lst*. Contiene expresiones temporales para determinar la franja horaria como (de la mañana, de la tarde, de la madrugada, etc).
- *timex_minutes.lst*. Contiene expresiones temporales en lenguaje natural para determinar los minutos de una determinada hora (y media, menos cuarto, y cuarto, en punto, etc).
- *timex_date.lst*. Contiene términos relacionados con fechas (años, días, etc.)
- *timex_fechasfuzzy.lst*. Contiene descripciones de fechas en lenguaje natural que tienen que traducirse a una determinada fecha (ayer, mañana, la próxima semana, etc).

Dentro de cada una de estas listas se incluye meta información para que se incluya dentro de las anotaciones *Lookup* obtenidas por el componente Gazetteer. Por ejemplo, en la Figura 39 se puede observar parte del fichero *timex_fechasfuzzy.lst*. Aquí, vemos que hay un conjunto de metadatos que representan la frecuencia con la que se repite la fecha (freq), la cantidad absoluta (quant), el modo si la fecha es anterior o posterior a la actual (mod) y el valor relativo de la fecha con respecto a la actual (value). Así, podemos ver que la expresión “ayer” tiene como mod=BEFORE y value=-1D porque representa que esa fecha es un día anterior al día actual. Otro ejemplo es la expresión “la semana

próxima" que tiene como mod=AFTER y value +1W que representa que hay que sumar una semana a la fecha actual.

III.3.3.1.3. Reglas JAPE para la detección de expresiones temporales

En este apartado se describen las expresiones regulares para la detección de expresiones temporales desarrolladas mediante reglas JAPE. Más concretamente estos ficheros son son los siguientes:

- general_fecha: Identifica distintos formatos de fecha a través de más de 25 reglas que incluyen la identificación de fechas completas, de fechas sin día, de fechas con sólo el año, etc.
- general_intervalofechas: Identifica distintos tipos de intervalos entre fechas.
- general_timesofday. Identifica distintos tipos de expresiones temporales de horas y minutos mediante la implementación de más de 10 reglas distintas que incluyen expresiones como "a las 9 de la mañana", "a las diez y media", etc.
- general_timesofdayinterval: Identifica distintos tipos de intervalos entre expresiones temporales horarias.

En la Figura 39 se muestra un fragmento del fichero GATE con las reglas JAPE que emplea el proceso *Transducer* para analizar fechas difusas.

```
ayer&type=DATE&freq=""&quant=""&mod=BEFORE&value=-1D
anteayer&type=DATE&freq=""&quant=""&mod=BEFORE&value=-2D
antes de ayer&type=DATE&freq=""&quant=""&mod=BEFORE&value=-2D
anoche&type=DATE&freq=""&quant=""&mod=BEFORE&value=-1D
durante el día de ayer&type=DATE&freq=""&quant=""&mod=BEFORE&value=-1D
durante todo el día de
ayer&type=DATE&freq=""&quant=""&mod=BEFORE&value=-1D
hace unos días&type=DATE&freq=""&quant=""&mod=BEFORE&value=-XD
hace días&type=DATE&freq=""&quant=""&mod=BEFORE&value=-XD
hoy&type=DATE&freq=""&quant=""&mod=AFTER&value=+0D
a lo largo del día&type=DATE&freq=""&quant=""&mod=AFTER&value=+0D
durante el día&type=DATE&freq=""&quant=""&mod=AFTER&value=+0D
mañana&type=DATE&freq=""&quant=""&mod=AFTER&value=+1D
pasado mañana&type=DATE&freq=""&quant=""&mod=AFTER&value=+2D
pasadomañana&type=DATE&freq=""&quant=""&mod=AFTER&value=+2D
dentro de unos días&type=DATE&freq=""&quant=""&mod=AFTER&value=+XD
dentro de días&type=DATE&freq=""&quant=""&mod=AFTER&value=+XD
```

```
los próximos días&type=DATE&freq=""&quant=""&mod=AFTER&value=+0D
en los próximos días&type=DATE&freq=""&quant=""&mod=AFTER&value=+0D
semana pasada&type=DATE&freq=""&quant=""&mod=BEFORE&value=-1W
pasada semana&type=DATE&freq=""&quant=""&mod=BEFORE&value=-1W
un día de la semana
pasada&type=DATE&freq=""&quant=""&mod=BEFORE&value=-1W
hace semanas&type=DATE&freq=""&quant=""&mod=BEFORE&value=-XW
hace algunas semanas&type=DATE&freq=""&quant=""&mod=BEFORE&value=-XW
la próxima semana&type=DATE&freq=""&quant=""&mod=AFTER&value=+1W
la semana próxima&type=DATE&freq=""&quant=""&mod=AFTER&value=+1W
la semana que viene&type=DATE&freq=""&quant=""&mod=AFTER&value=+1W
un día de la semana
próxima&type=DATE&freq=""&quant=""&mod=AFTER&value=+1W
un día de la semana que
viene&type=DATE&freq=""&quant=""&mod=AFTER&value=+1W
un día de la próxima
semana&type=DATE&freq=""&quant=""&mod=AFTER&value=+1W
las próximas semanas&type=DATE&freq=""&quant=""&mod=AFTER&value=+XW
las semanas próximas&type=DATE&freq=""&quant=""&mod=AFTER&value=+XW
las semanas que vienen&type=DATE&freq=""&quant=""&mod=AFTER&value=+XW
```

Figura 39. Extracto del fichero timex_fechasfuzzy.lst.

A continuación se explican brevemente los ficheros de reglas general_fecha.jape y general_timesofday.jape que son la parte central de la detección de expresiones temporales.

III.3.3.1.3.1 Reglas para obtención de fechas (general_fecha)

El fichero general_fecha.jape contiene más de 25 reglas distintas y en este apartado se van a comentar algunas de ellas. En la Figura 40 se puede observar parte del contenido de las reglas para detectar fechas. Como se puede observar, se muestran cuatro reglas para obtener fechas utilizando las anotaciones *Lookup* obtenidas del Gazetteer y las anotaciones *Token y NewLineToken* obtenidas del Tokeniser.

La primera regla identifica como entidad *Fecha*, todas las fechas completas formadas por día, mes y año escritas en lenguaje natural que incluye las expresiones siguientes:

- el DIA_SEMANA , día NÚMERO de MES de NÚMERO
    - El lunes día 24 de diciembre de 2012
    - El lunes día 31 de diciembre del 12

- DIA_SEMANA (,) NÚMERO de MES de NÚMERO
    - Lunes 24 de diciembre de 2012
    - Lunes 31 de diciembre del 12
- NÚMERO de MES de NÚMERO
    - 25 de diciembre de 2012
    - 31 de diciembre del 12

La segunda regla identifica como entidad *Fecha*, todas las fechas formadas por día y mes escritas en lenguaje natural. Es decir las fechas que no incluyen el año y que abarcan las expresiones siguientes:

- el DIA_SEMANA , día NÚMERO de MES
    - El lunes día 24 de diciembre
    - El lunes día 31 de diciembre
- DIA_SEMANA (,) NÚMERO de MES
    - Lunes 24 de diciembre
    - Lunes 31 de diciembre
- NÚMERO de MES de NÚMERO
    - 25 de diciembre
    - 31 de diciembre

La tercera regla identifica como entidad *Fecha*, toda las fechas formadas por mes y año escritas en lenguaje natural. Es decir las fechas que no incluyen el día y que abarcan principalmente las expresiones siguientes:

- MES de NÚMERO
    - diciembre de 2012
    - enero del 13

La última regla de ejemplo identifica como entidad *Fecha*, toda las fechas formadas solamente por el mes, que identifica solamente la expresión siguiente:

- MES
    - Diciembre
    - Enero

```
Phase:      general_fecha
Input:  Token Lookup NewLineToken
```

```
Options: control = appelt

// Fecha rules
Rule:FechaCompleta1
Priority: 80
(
  ({Token.string=="el"})?
  ({Lookup.minorType == dayoftheweek}
  ({Token.string == ","}) ?
  ({Token.lemma == "día"})?)?
  ({Token.kind == number} | {Lookup.majorType == number}):day
  {Token.string == "de", !NewLineToken}
  ({Lookup.minorType == month, !NewLineToken}):month
  ({Token.string == "de", !NewLineToken}|{Token.string ==
"del", !NewLineToken})
  (({Token.kind == number, Token.length == 2}) |
  ({Token.kind == number, Token.length == 4})):year
)
:fecha -->
...

Rule:FechaSinAnyo
Priority: 60
(
  ({Token.string=="el"})?
  ({Lookup.minorType == dayoftheweek}
  ({Token.string == ","}) ?
  ({Token.lemma == "día"})?)?
  ({Token.kind == number} | {Lookup.majorType == number}):day
  {Token.string == "de", !NewLineToken}
  ({Token.kind == number} | {Lookup.minorType ==
month, !NewLineToken}):month
)
:fecha -->
...

Rule:FechaSinDia
Priority: 50
(
  ({Lookup.minorType == month, !NewLineToken}):month
  {Token.string == "de", !NewLineToken}
```

```
  (({Token.kind == number, Token.length == 2}) |
  ({Token.kind == number, Token.length == 4})):year
)
:fecha -->
...

Rule:FechaSoloMes
Priority: 35
(
  ({Lookup.minorType == month}):month
)
:fecha -->
...
```

Figura 40 Parte del fichero general_fecha.jape desarrollado.

III.3.3.1.3.2 Reglas para obtención de tiempos (general_timesofday)

El fichero general_timesofaday.jape contiene más de 10 reglas distintas para identificar distintos tiempos a lo largo del día y en este apartado se van a comentar algunas de ellas. En la Figura 41 se puede observar parte del contenido de las reglas para detectar fechas. Como se puede observar, se muestran tres reglas de ejemplo para obtener tiempos utilizando las anotaciones *Lookup* obtenidas del Gazetteer y las anotaciones *Token y NewLineToken* obtenidas del Tokeniser y las anotaciones *Application* que contienen el tipo de aplicación a la que se va a aplicar el comando verbal.

La primera regla identifica como entidad *Time*, todas las expresiones que representan tiempos compuestos solamente por la hora escritos en lenguaje natural que incluye principalmente la expresión siguiente:

- (a) la(s) NUM (horas) (MOMENTO_DEL_DIA):
    - A la 1 del mediodía
    - A las 3 horas de la tarde
    - A las 9 a.m.
    - A las 10.
    - 1 de la madrugada

La segunda regla identifica como entidad *Time*, todas las expresiones que representan tiempos compuestos por la hora y los minutos escritos en lenguaje natural que abarcan principalmente las siguientes expresiones:

- (a) la(s) NUM (horas) y NUM (minutos) (MOMENTO_DEL_DIA):
    - A las 9 horas y 30 minutos de la mañana
    - A las 3 y 15 minutos de la tarde
    - A las 9 y 30 minutos a.m.
    - A las 9 y 30
- (a) la(s) NUM (horas) MINUTOS_FUZZY (MOMENTO_DEL_DIA):
    - A las 9 y media de la mañana
    - A las 3 y cuarto de la tarde
    - A las 9 menos cuarto de la tarde.
    - A las 12 en punto.
- NUM (:|.|y) NUM (MOMENTO_DEL_DIA):
    - 9 y 30 esta tarde
    - 9:30
    - 9.30

La última regla identifica como entidad *Time*, todas las expresiones que representan tiempos compuestos por la hora y los minutos escritos en lenguaje natural, pero en el que el momento del día se diría al inicio. Esta regla identifican principalmente las siguientes expresiones:

- MOMENTO_DEL_DIA (a) la(s) NUM (horas) y NUM (minutos):
    - Por la mañana a las 9 horas y 30 minutos
    - Por la tarde a las 3 y 15 minutos
- MOMENTO_DEL_DIA (a) la(s) NUM (horas) MINUTOS_FUZZY:
    - Por la mañana a las 9 y media
    - Por la tarde a las 3 en punto.

- MOMENTO_DEL_DIA NUM (:|.|y) NUM:
    - Tarde 9 y 30
    - Por la mañana 9:30

```
  Phase:     general_timesofday
  Input:  Token Lookup NewLineToken Application
  Options: control = applet

  Rule:Time1 // [a] la[s] HORA MOMENTO_DEL_DIA
  Priority: 50
  (
    (
      (HOUR_CONTEXT)?
      (HOUR):hour
      (HOUR_SUFFIX)?
      ({Lookup.minorType == timesofday_post} | {Lookup.minorType ==
timesofday}):timesofday
    ):timex
  ):time -->TIME_CONVERTER
  // Ejemplos Time1:
  //   " a la 1 del mediodía"
  //   "a las 1 de la noche"
  //   "a las 5 de la tarde"
  //   "a las 11 a.m."

  Rule:Time2 // [[a] la[s]] HORA [horas | ...] [y | . | :] MIN [minutos | min
| ...] [MOMENTO_DEL_DIA]
  Priority: 50
  (
    (
      (HOUR_CONTEXT)?
      (HOUR):hour
      (HOUR_SUFFIX)?
      (TIME_PUNCTUATION)?
      (MIN):min
      (MIN_SUFFIX)?
      ({Lookup.minorType == timesofday_post} | {Lookup.minorType ==
timesofday})?:timesofday
    ):timex
  ):time -->TIME_CONVERTER
  // Ejemplos Time2:
  //   "a las 9.25 am"
  //   "las 9 horas y 25 minutos de la tarde"
  //   "21 horas 25 minutos"
  //   "las 21.25 de la noche"

  Rule:Time3 // [MOMENTO_DEL_DIA] [[a] la[s]] HORA [horas | ...] [y | . | :]
MIN [minutos | min | ...]
  Priority: 50
  (
```

```
    (
       ({Lookup.minorType == timesofday_pre} | {Lookup.minorType ==
timesofday})?:timesofday
       (HOUR_CONTEXT)?
       (HOUR):hour
       (HOUR_SUFFIX)?
       (TIME_PUNCTUATION)?
       (MIN):min
       (MIN_SUFFIX)?
    ):timex
  ):time -->TIME_CONVERTER
  // Ejemplos Time3:
  //   "por la mañana a las 9.25 am"
  //   "por la tarde a las 9 horas 25 minutos"
  //   "esta tarde a las 9 horas y 25 minutos"
```

Figura 41 Parte del fichero general_timesofday.jape desarrollado.

III.3.3.1.4. Unidad de estimación y resolución de expresiones temporales

Una vez las expresiones temporales han sido identificadas, deben ser resultas. Las anotaciones obtenidas en el módulo anterior tienen información y datos relativos para el cálculo de la fecha y hora global. En la Figura 42 se muestra un ejemplo de anotaciones resultantes del módulo anterior.

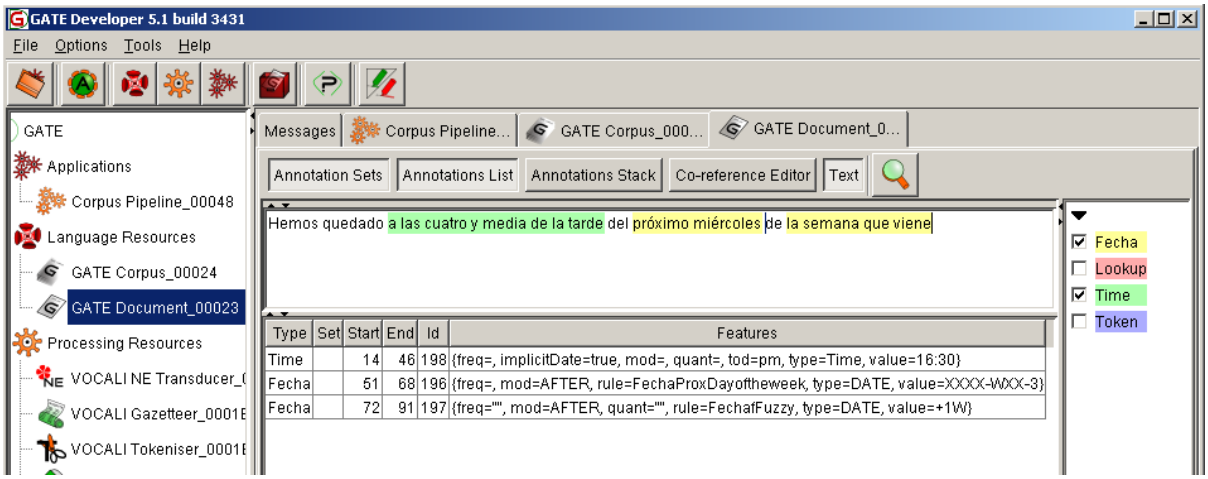


Figura 42. Anotaciones temporales con información relativa sobre fechas y hora.

Como se puede observar en la Figura 42, aunque la hora está bien calculada como 16:30, los datos sobre la fecha de la cita están descritos de manera relativa. Por ejemplo, “próximo miércoles” se refiere al día tres de la semana después del día actual. Esto es, si estamos de miércoles a domingo se refiere al miércoles de la semana siguiente, pero si estamos a lunes a martes, se refiere el miércoles de esa misma semana. Por último, la expresión "la semana que viene" hace referencia a una semana más de la actual, con lo

que juntando las dos anotaciones el sistema debería inferir que se refiere al tercer día de la semana siguiente a la actual.

Para la resolución de estas expresiones implícitas se utiliza un conjunto de heurísticas que junto con los datos relativos obtenidos en las anotaciones obtienen una fecha y una hora absolutas.

Las clases software que implementan esta funcionalidad son las siguientes:

- GateDateResolver.java: Implementa distintos métodos para resolver las anotaciones tipo *Fecha* que no son absolutas.
- GateTimeResolver.java: Implementa distintos métodos para resolver las anotaciones tipo *Time* que no son absolutas.
- GateTimexResolver.java: Implementa distintos métodos para la integración de las anotaciones *Fecha* y *Time* en una fecha y tiempo absoluto.
- ResolveGateTimexUtil.java: Implementa distintos tipos de utilidades utilizadas por las clases anteriores.

Estás clases están dentro del paquete nlp/gate/timex de la aplicación global del proyecto.

Por otro lado es de sobra conocido que la implementación de las clases Java para el manejo de fechas y horas tiene una funcionalidad muy limitada además de numerosos bugs. Por lo tanto se ha optado por trabajar con la librería JODA-TIME para la manipulación de fechas y horas, aunque sólo para la manipulación, mientras que se seguirá usando la clase Date para representar horas y fechas entre los distintos módulos del sistema por su compatibilidad con JAXB, que es la librería elegida para generar los comandos XML.

III.3.4. RESOLUCIÓN DE AMBIGÜEDADES

En este apartado del módilo NLP se detalla el sistema de resolución de ambigüedades entre las anotaciones y entidades nombradas extraídas en otros módulos. El objetivo principal de este módulo es resolver las posibles ambigüedades derivadas de la naturaleza del lenguaje natural que puedan confundir al sistema en la detección del comando a realizar. Por ejemplo, el comando "recuérdame que tengo que enviar un correo electrónico a Antonio diciéndole que ya está reservado el hotel para las

vacaciones" está asociado a la creación de una nueva tarea en la aplicación del móvil, no al envío de un correo electrónico.

Además, es muy probable que una misma palabra pueda representar varios significados. En nuestro caso la palabra "correo" puede significar o bien la aplicación del correo electrónico o bien hacer referencia a los datos de un correo de una persona. Otras palabras como "Murcia" pueden ser a su vez una localización o bien un apellido de una persona.

Para resolver este objetivo primero se realizó un estudio para poder determinar posibles ambigüedades del lenguaje en el dominio de la tesis doctoral para, una vez detectadas, se han generado unas reglas lógicas que permitan desambiguar los comandos procesados. Este sistema de reglas se ha implementado en JAVA mediante GATE y permite desambigüar y obtener el sentido correcto de las palabras dentro de un texto.

Por ejemplo, en la figura siguiente (Figura 43) podemos ver un comando en lenguaje natural donde existen varias ambigüedades que es necesario resolver en este sistema. Como se puede observar en la figura, las palabras "Murcia" y "Madrid" se etiquetan como localizaciones pero forman parte del nombre de una persona. También se puede observar que la palabra "email" se identifica como una aplicación y como un dato.

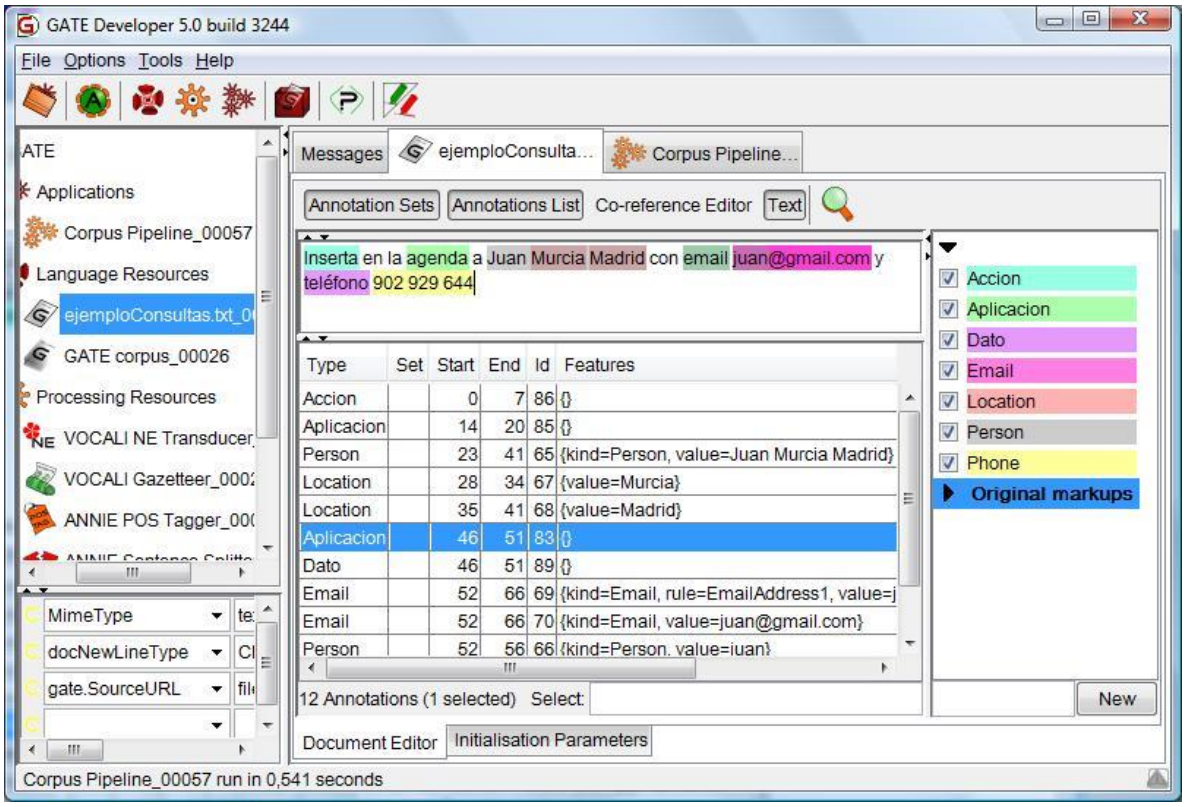


Figura 43. Ejemplo de ambigüedad.

Para la realización de este módulo se evaluaron diferentes trabajos de desambiguación conceptual que se resumen a continuación y se presentan en (Vázquez Pérez, 2009).

III.3.4.1.1. Clasificación de sistemas de desambiguación de palabras

Desde los primeros sistemas hasta la actualidad han surgido nuevas propuestas y distintos enfoques para resolver este problema. Una forma muy extendida de clasificar los sistemas de desambiguación se basa en la principal fuente de conocimiento utilizada para establecer los distintos sentidos que puede tomar dicho término. Primero, existen los métodos que utilizan diccionarios, tesauros, bases de conocimiento léxicas como ontologías en lenguaje natural, y reglas de desambiguación sin utilizar ningún corpus. Este tipo de métodos se denominan basados en diccionarios o basados en conocimiento. Por otro lado, existen métodos que evitan tener información externa y trabajan con corpus sin etiquetar. Éstos se suelen denominar métodos no supervisados. Por último, existen los sistemas supervisados y semi-supervisados, que utilizan corpus anotados semánticamente como entrenamiento, o como fuente de datos en un sistema.

Casi todas las aproximaciones de aprendizaje supervisado se han aplicado a la desambiguación, incluyendo algoritmos agregativos y discriminativos y técnicas asociativas tales como selección de características, optimización de parámetros.

Los métodos no supervisados tienen la ventaja de evitar el problema existente en la adquisición y formalización de nuevo conocimiento. Estos métodos son capaces de inferir el sentido del término, a partir de textos de entrenamiento, agrupando (mediante clusters) ocurrencias de palabras y clasificando entonces nuevas ocurrencias en los clusters inducidos.

Las propuestas basadas en conocimiento de los años 80 están todavía en proceso de investigación y han obtenido muy buenos resultados. Las principales técnicas utilizan restricciones de selección, el solapamiento de textos, medidas de similitud semántica y reglas de desambiguación. Actualmente, la tendencia es hacer una inferencia semántica general utilizando bases de conocimiento, y reglas de desambiguación obteniendo como resultado el sentido correcto del término.

De los distintos tipos de métodos explicados el que más interesa es el basado en conocimiento que se explica con más detalle a continuación.

III.3.4.1.2. Métodos basados en conocimiento

En esta categoría encontramos diferentes algoritmos para la extracción automática de los sentidos de los términos que puedan presentar una ambigüedad. Normalmente, el rendimiento de estos métodos basados en conocimiento, es muy bueno y suelen tener una amplia cobertura ya que pueden aplicarse a cualquier tipo de texto en comparación con las otras aproximaciones que solamente se pueden aplicar a aquellas palabras de las que se dispone de corpus anotados.

A continuación vamos a enumerar diferentes técnicas utilizadas por los métodos basados en conocimiento, aplicables sobre cualquier base de conocimiento léxica que defina sentidos de palabras y relaciones entre ellas. La base de conocimiento léxica más utilizada es WordNet (Miller, 1995). Se pueden distinguir cuatro tipos principales de métodos basados en conocimiento:

1. El algoritmo de Lesk (Lesk, 1986), en el cual, los sentidos de las palabras de un contexto se identifican basándose en una medida de solapamiento contextual entre las definiciones de un diccionario.
2. Medidas de similitud semántica extraídas a través de redes semánticas. Esta categoría incluye métodos que tratan de encontrar la distancia semántica existente entre diferentes conceptos. Dependiendo del tamaño del contexto estas medidas se dividen en dos grandes categorías:
    a. Métodos aplicables a un contexto local, donde las medidas de similitud semántica se utilizan para desambiguar palabras conectadas por relaciones sintácticas o por su localización.
    b. Métodos aplicables a contextos globales, donde las cadenas léxicas son derivadas basándose en medidas de similitud semántica (una cadena léxica es un hilo de significado extraído a través del texto total).
3. Preferencias de selección adquiridas de forma automática o semi-automática, como una forma de restringir los posibles sentidos de una palabra, basados en la relación que ésta tiene con otras palabras en el contexto.

4. Métodos heurísticos, que consisten en reglas que pueden asignar un sentido a ciertas categorías de palabras, incluyendo:
    a. El sentido más frecuente.
    b. Un sentido por colocación.
    c. Un sentido por discurso.

En este trabajo se ha optado por el desarrollo de un método heurístico que permita desambiguar los conceptos encontrados en base a unas reglas de desambiguación implementadas en JAPE dentro del framework GATE. A continuación se describen en detalle los distintos tipos de trabajos basados en heurística.

III.3.4.1.2.1 Heurísticas para desambiguación

Una forma sencilla de establecer el sentido correcto de los términos en un texto es utilizar heurísticas basadas en propiedades lingüísticas aprendidas a través de textos. Una de las heurísticas más utilizadas como base es la denominada "sentido más frecuente".

Además de esta heurística, existen otras dos comúnmente utilizadas cuya base es la suposición de que una palabra siempre tiene el mismo sentido en: todas sus ocurrencias en un mismo discurso ("un sentido por discurso") o en la misma posición ("un sentido por colocación") o en el mismo dominio.

Sentido más frecuente

Entre todos los posibles sentidos que puede tener una palabra, generalmente existe uno que ocurre más a menudo que los otros sentidos. Por lo tanto, un sistema muy simple de desambiguación sería aquel que asignara a cada palabra su sentido más frecuente.

Aunque conceptualmente es muy sencillo, y casi trivial de implementar, hay un inconveniente asociado a este método: no siempre disponemos de la distribución de las ocurrencias de los sentidos en todos los idiomas, ya que, no existen suficientes textos disponibles para extraer esa distribución. Además, un cambio en el dominio generalmente altera la distribución de los sentidos, disminuyendo así los resultados obtenidos por esta heurística.

Un sentido por posición

Esta heurística tiene una hipótesis similar a la heurística de un sentido por discurso, pero aplicada en un ámbito diferente. Supone que una palabra tiende a tener el mismo sentido cuando se utiliza en la misma posición. Es decir, las palabras cercanas dan pistas acerca del sentido de una palabra. Además, se ha determinado que este efecto es mayor para posiciones adyacentes y empieza a decrecer cuando la distancia entre palabras aumenta.

Se han desarrollado distintos experimentos con palabras con sentidos bien diferenciados y con sentidos muy próximos entre sí. Al igual que en el caso anterior, los resultados empeoran cuando se consideran palabras con sentidos con diferencias sutiles.

Un sentido por discurso.

Esta heurística se basa en que una palabra tiende a preservar su sentido a través de todas sus ocurrencias en un discurso determinado. Esta medida permite establecer el sentido de una misma palabra identificándolo una única vez.

Esta heurística funciona bien con palabras que tienen sentidos bien diferenciados. En el caso en que tengamos palabras con sentidos con una diferencia muy sutil, este método obtiene peores resultados. Se ha demostrado que palabras polisémicas con sentidos muy similares, pueden tener más de un sentido por discurso.

III.3.4.2. Arquitectura de la solución

La solución planteada se basa se basa en el trabajo presentado en (Vázquez Pérez, 2009) y modela un sistema basado en conocimiento, reglas y heurísticas para la desambiguación usando la infraestructura GATE y en particular usando reglas JAPE que son apropiadas para detectar, y posteriormente anotar, las expresiones ambiguas y en base al contexto poder decidir qué sentido será el correcto en el comando del dispositivo móvil.

Para ello, se ha desarrollado un modelo de desambiguación, basado en heurísticas para poder apoyar las reglas JAPE definidas. Dicho módulo se ha desarrollado un módulo software en JAVA que permite desambiguar dentro del dominio de aplicación de este trabajo.

La arquitectura funcional del módulo de desambiguación se puede ver en la Figura 44.

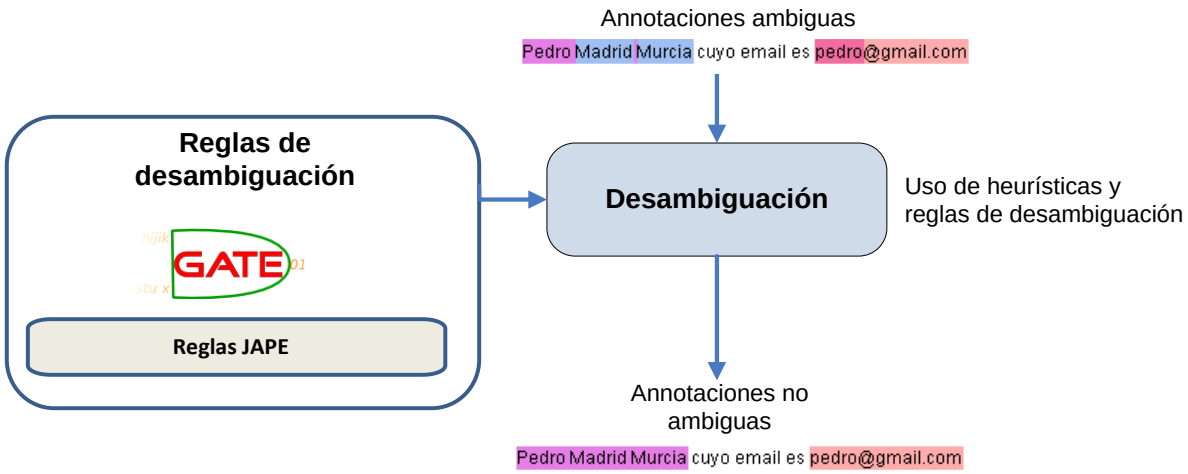


Figura 44. Arquitectura del sistema de desambiguación.

Este sistema toma como entrada todas las anotaciones obtenidas por GATE y en base al módulo de desambiguación que se apoya en distintas reglas JAPE definidas, se obtiene otro conjunto de anotaciones no ambiguas que puedan representar uno o varios comandos para un dispositivo móvil.

A continuación se describen algunas de las heurísticas implementadas.

III.3.4.2.1. Heurística: la anotación más grande tiene prioridad

Esta heurística dará prioridad a la anotación más grande en el caso de haber una ambigüedad en dichas anotaciones. Con esta regla, si una anotación se encuentra contenida en otra anotación, prevalecerá la anotación más grande. Veamos cómo funciona esta heurística con unos ejemplos.

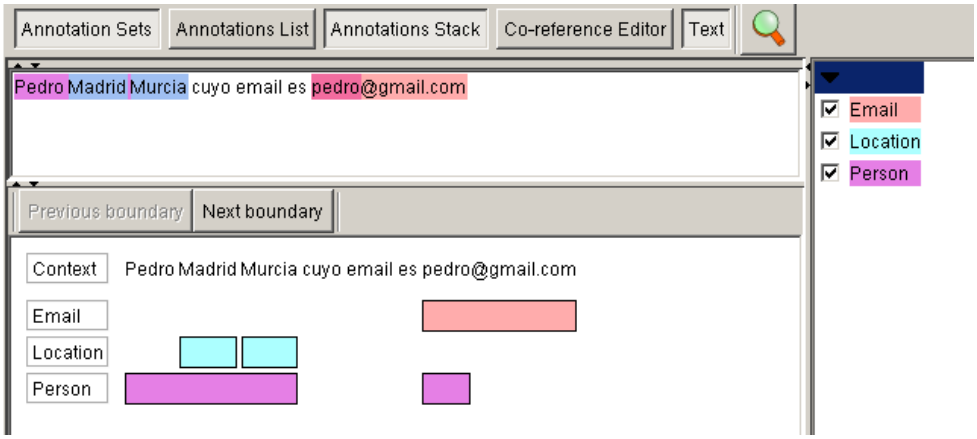


Figura 45. Ejemplo de ambigüedades donde una anotación está incluida dentro de otra.

En la Figura 45 se pueden ver distintas anotaciones ambiguas. Por un lado, los términos “Madrid” y “Murcia” se han etiquetado como *Location* pero además forman parte de una entidad *Person* “Pedro Madrid Murcia”. Como las anotaciones de *Location* están incluidas en la anotación *Person*, éstas se eliminarían dejando ese fragmento de texto sin ambigüedad. Por otro lado, una anotación “pedro” de tipo *Person* se solapa con otra anotación “pedro@gmail.com” de tipo *Email*. Como antes, la anotación de tipo *Person* se eliminaría dejando únicamente la anotación más grande de tipo *Email*. El resultado de la desambiguación puede verse en la Figura 46.

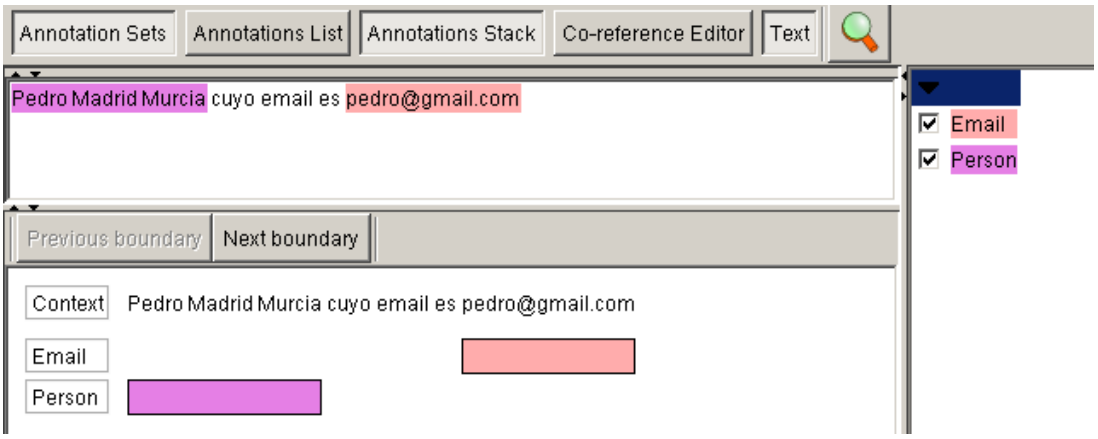


Figura 46. Ejemplo de resolución de ambigüedades.

III.3.4.2.2. Heurística para reglas relacionadas con la posición de la aparición en la orden.

Las anotaciones que tienen que ver con acciones (*Action*) como insertar, eliminar, enviar consultar, etc. y las anotaciones que tienen que ver con las aplicaciones (*Application*) como correo electrónico, sms, agenda, etc., no se tienen en cuenta a partir de un número de caracteres que es configurable.

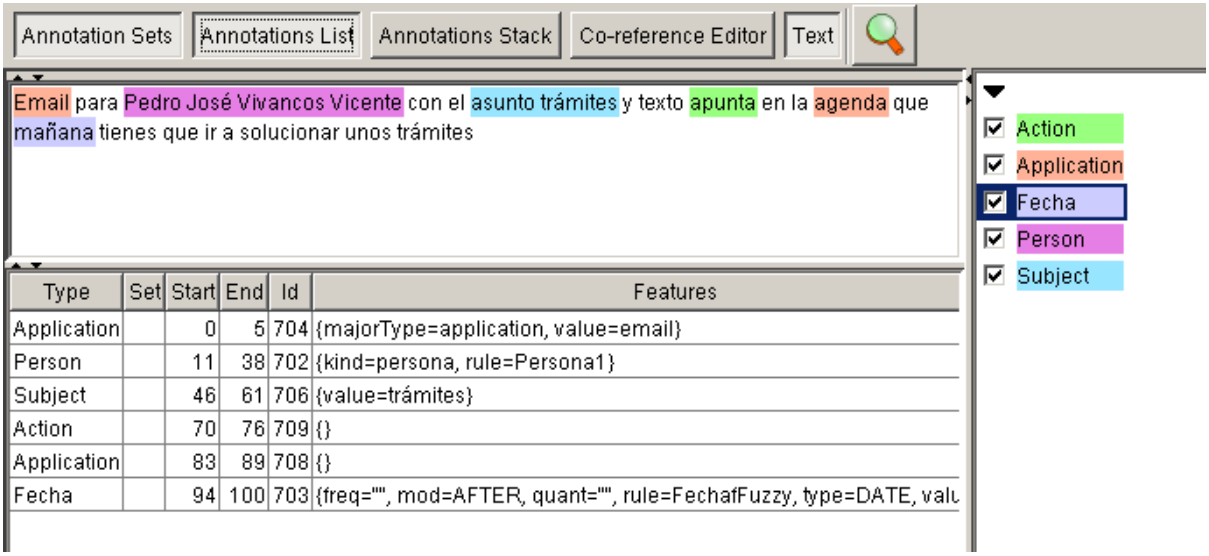


Figura 47. Ejemplo de ambigüedades donde existen anotaciones de tipo Action y Application en posiciones avanzadas del comando.

Como se puede ver en la Figura 47, pueden aparecer anotaciones de tipo *Action* como "apunta" o tipo *Application* como "agenda", pero se puede ver que en el contexto no se está haciendo referencia a eso, sino que forman parte del email que se le tiene que enviar a la persona identificada. Si aplicamos esta heurística se eliminarán estas anotaciones a partir de la mitad del mensaje para no producir errores. En la Figura 48 se muestra cómo quedarían las anotaciones definitivas.

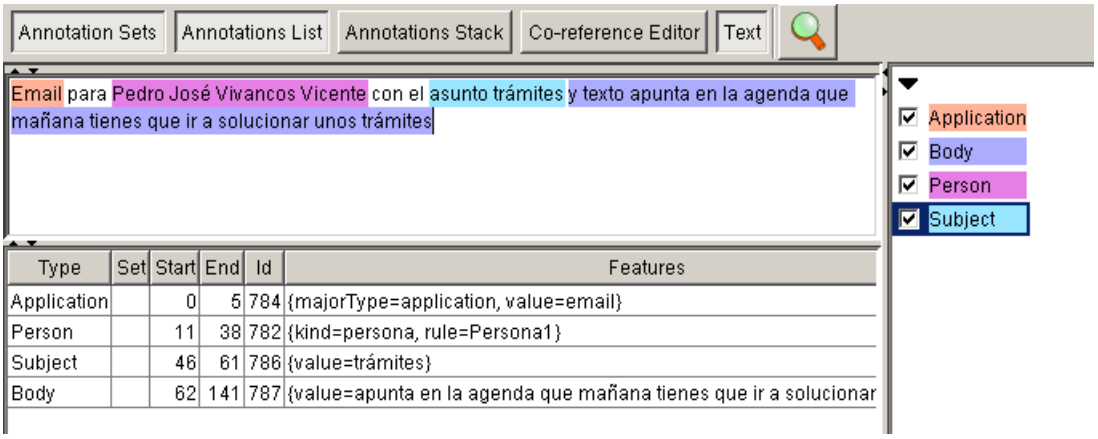


Figura 48. Ejemplo de eliminación de anotaciones de tipo Action y Application.

III.3.4.2.3. Heurísticas relacionadas con la detección de la entidad Application

Una de las cosas más importantes es detectar la aplicación sobre la cual se tiene que ejecutar el comando en lenguaje natural. Es probable que durante el documento puedan haber distintas anotaciones que representen las aplicaciones pero cada comando sólo puede tener asociada una única aplicación. Por ejemplo, en la Figura 49 se muestra un caso donde aparecen dos anotaciones de tipo *Application*, "email" y "agenda". Si

observamos el ejemplo queda claro que la aplicación sobre la que se tiene que realizar dicha orden es la agenda y no el email.

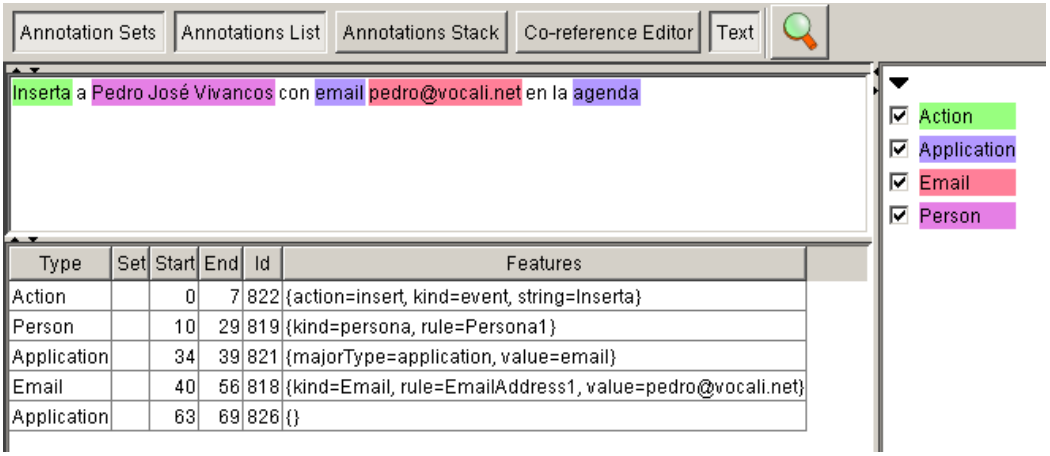


Figura 49. Ejemplo de ambigüedad de la entidad Application.

Para poder obtener y quedarnos con sólo una aplicación que sea la correcta se ha implementado un algoritmo con un conjunto de heurísticas que se muestran a continuación:

1. Si existe sólo una anotación de tipo Aplication no hay ambigüedad y se termina el algoritmo.
2. Si no existe ninguna o existen varias anotaciones de tipo Aplication:
   a. Si no existe hora pero si fechas: desambiguar como Aplication.type = Calendar y finalizar
   b. Si existen varias fechas o si existen varias horas: desambiguar como Aplication.type = Calendar y finalizar
   c. Si existe sólo una fecha y una hora:
      i. Si tiene una anotación que represente a una persona o lugar: desambiguar como Aplication.type = Calendar y finalizar.
      ii. Si no, no se puede resolver.
   d. Si no existen fechas ni horas:
      i. Pero existe una persona: desambiguar a la agenda de contactos y finalizar.
      ii. Si no, no se puede resolver.

III.3.4.2.4. Heurísticas relacionadas con la detección de la entidad Action

Es muy probable que los usuarios no digan la acción a realizar en algunos comandos sobre distintas aplicaciones del terminal móvil. Por ejemplo en la Figura 50 aparece un comando donde no se indica la acción a realizar. De esta figura se puede deducir que la acción que se pretende realizar es "enviar".

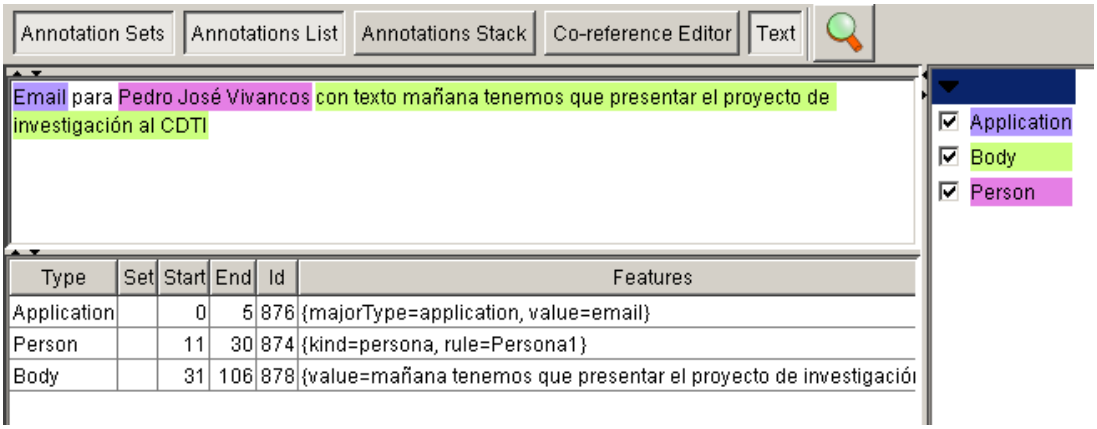


Figura 50. Ejemplo de comando sin entidad de tipo *Action*.

Para poder obtener y quedarnos con la acción correcta se ha implementado un algoritmo con un conjunto de heurísticas de las cuales unas cuantas se muestran a continuación:

- Si no existe una anotación de tipo Action:
    - Si la aplicación es el calendario o la alarma: se crea una anotación Action = "Insert" y finalizar.
    - Si la aplicación es la agenda de contactos y se tiene el nombre, email y teléfono de la persona: se crea una anotación Action = "Insert" y finalizar.
    - En otro caso, lanzar excepción.
- Si existen varias anotaciones de tipo Action distintas: lanzar excepción.

III.3.5. RESULTADO MOTOR NLP: COMANDO ESTRUCTURADO

Este nuevo apartado presenta y describe el modelo de datos que representan a los comandos estructurados que el servicio web a desarrollar en el trabajo emitirá para que las correspondientes aplicaciones de las distintas plataformas móviles ejecuten.

El prototipo busca ser capaz de controlar distintos elementos del móvil a través de la voz en lenguaje natural. Concretamente, el sistema debe ser capaz de gestionar el calendario de eventos, la agenda de contactos y el sistema de alarma/despertador.

El modo de funcionamiento es el siguiente: el usuario da un orden vocal que el teléfono móvil registra y, usando el servidor ASR, se transcribe a texto almacenándose en formato de nuevo SMS. El terminal móvil dispondrá de una aplicación que se encargará de leer ese texto y enviarlo a un servicio web que se encargará de procesar dicho texto, clasificar a qué tipo de acción se refiere (si acaso lo es de alguna de las indicadas al principio de este documento) y devolver al terminal móvil el comando en un formato estructurado que éste usará para modificar el calendario, la agenda o la alarma según la información obtenida.

III.3.5.1. Análisis de comandos soportados

En este apartado se presenta el XML Schema propuesto en esta tesis doctoral. En este modelado se representan las distintas acciones que se pueden efectuar sobre los elementos del móvil comentados anteriormente: la agenda de contactos, el calendario de eventos y el sistema de alarma/despertador.

III.3.5.1.1. Consideraciones previas

En la elaboración de estos comandos se ha intentado contemplar el mayor número posible de alternativas, sin embargo, se considera que debido a la escasez inicial de recursos y para abordar el problema con la mayor brevedad posible, lo interesante sería seleccionar algunas de estas acciones para ser implementadas, seleccionando únicamente aquellas que demuestren la viabilidad y potencia del prototipo.

III.3.5.1.2. Salida del sistema

La salida generada por este módulo consistirá en un mensaje XML que contendrá toda la información necesaria para ejecutar el comando de una forma estructurada y sin ambigüedades. Para ello, se ha definido en este trabajo un esquema XML (XSD) que estructura la información generada ante el procesamiento de cada comando.

Ejempos de esta salida así como el fichero XML Schema creado se presentan en "Anexo II. Comandos de voz implementados".

III.4. APLICACIÓN MÓVIL

III.4.1. INTRODUCCIÓN

III.4.1.1. Propósito

Este nuevo apartado recoge recoge todos los requisitos funcionales y no funcionales de las aplicaciones móviles que permitirán el uso del sistema de control de las funcionalidades del teléfono mediante comandos en lenguaje natural.

III.4.1.2. Ámbito

En esta sección se describen los requisitos de la aplicación que se ejecutará en los dispositivos móviles y que llamaremos en adelante Aplicación Móvil.

La Aplicación Móvil será el medio a través del cual el usuario podrá acceder a un conjunto de funciones de su dispositivo móvil mediante comandos en lenguaje natural. La Aplicación Móvil será la encargada de enviar el comando al servicio de procesamiento, recoger la acción a ejecutar y llevarla a cabo.

La Aplicación Móvil, facilitará así a todo tipo de personas el acceso a las funciones del dispositivo móvil, especialmente a personas con dificultades para el manejo de estos dispositivos.

III.4.2. DESCRIPCIÓN GENERAL DE LA APLICACIÓN

III.4.2.1. Perspectiva del producto

La Aplicación Móvil se integra dentro del sistema general tal y como se mostraba en la Figura 19: el usuario da órdenes al teléfono, que simula que lo entiende utilizando para ello el servicio de procesamiento del lenguaje compuesto por el servidor ASR y el servidor NLP.

III.4.2.2. Interfaces de usuario

El usuario interactuará con la aplicación fundamentalmente mediante la voz, que utilizará para dar los comandos que quiere llevar a cabo.

III.4.2.3. Interfaces de sistema

La Aplicación Móvil se comunicará con los otros componentes del sistema a través de Internet utilizando los protocolos de comunicación que se describen a lo largo de este apartado.

III.4.3. FUNCIONES DE LA APLICACIÓN

Se distingue entre dos tipos de funciones:

1. **Funciones bajo demanda**: el usuario solicita su ejecución mediante un comando de voz y el dispositivo móvil las lleva a cabo. Son el tipo de funciones principales en este trabajo.
2. **Funciones activas**: este tipo de funciones son iniciadas por la Aplicación Móvil y no por el usuario en respuesta a determinados eventos como una llamada entrante o un nivel de batería bajo.

Funciones bajo demanda:

1. Llamar a un teléfono.
2. Introducir un nuevo contacto.
3. Introducir un nuevo evento en la agenda.
4. Enviar un correo electrónico.
5. Enviar un SMS.
6. Introducir una nota.
7. Introducir una tarea.
8. Establecer una alarma.
9. Seleccionar perfil de sonido (silencio, vibración, etc.)
10. Consulta de llamadas perdidas.
11. Consulta de eventos en la agenda.
12. Emergencia.

Funciones activas:

1. Atender llamada entrante.
2. Notificación de batería baja.
3. Sistema de alerta de accidentes.

III.4.3.1. Características de los usuarios

Se pretende que la Aplicación Móvil sea usada por un variado abanico de usuarios, desde personas mayores o con restricciones de movilidad hasta usuarios para los que la interacción por voz puede suponer una ventaja, como personas que utilizan el móvil conduciendo o en situaciones donde tienen las manos ocupadas.

Por tanto, las características de los usuarios en cuanto a edad, nivel de uso de la tecnología, etc. será muy variada y la Aplicación Móvil tendrá que ser adecuada por igual para todos.

III.4.4. REQUISITOS ESPECÍFICOS

III.4.4.1. Requisitos funcionales

Describiremos en este apartado cada una de las funciones que ha de implementar la Aplicación Móvil.

III.4.4.1.1. Funciones bajo demanda

III.4.4.1.1.1 Llamar a un teléfono

Establece la llamada a un destino, el cual se puede introducir directamente como número de teléfono, o puede pertenecer a un contacto de la agenda. En este último caso, por defecto llama al destino móvil, aunque se ofrece la posibilidad de hablar con el teléfono memorizado como tipo casa o trabajo.

Ejemplos:

- Hablar con Joaquín Rodríguez.
- Llamada a la casa de Pedro Nortes.
- Marca el 555 996 215.

III.4.4.1.1.2 Nuevo contacto

Permite crear un nuevo contacto en el móvil (nombre, apellidos, teléfono y correo electrónico).

Ejemplos:

- Crea el nuevo contacto Pedro García con teléfono 555 125 321.
- Nuevo contacto Antonio López. Teléfono 555 146 846 y correo alopez@acme.org.
- Inserta un contacto con el nombre de Laura Gómez y número 555 789 654.

III.4.4.1.1.3 Nuevo evento en la agenda

Registra un nuevo evento en el calendario indicando el comienzo y fin, lugar, asistentes, etc.

Ejemplos:

- Inserta en la agenda una cita con Pedro Vivancos el próximo lunes por la mañana.
- Nuevo evento en el calendario reunión a las 5 de la tarde del próximo martes con Rafael Valencia en las Oficinas.
- Añade una cita con Juan Pérez el próximo día 15 a las 9 de la mañana en la Universidad.

III.4.4.1.1.4 Dictar y enviar un correo electrónico

Genera un correo electrónico y lo envía a uno o varios destinatarios. La opción de incluir los destinatarios se puede dictar en formato contacto de la agenda o directamente dirección de email. En caso de que no se especifique destinatario/os, el email se quedaría almacenado como borrador.

Ejemplo:

- Crear un nuevo correo electrónico para enviar a José Ruiz y a d.garcia@neosistec.com, con asunto Encuesta INVOX y texto por favor, sea tan amable de concedernos unos minutos para rellenar la siguiente encuesta con el fin de mejorar la calidad de nuestros productos y poder ofrecerle un mejor servicio. Gracias.

III.4.4.1.1.5 Dictar y enviar un mensaje sms

Genera un mensaje sms y lo envía a uno o varios destinatarios, ya sean contactos de la agenda o se indique directamente los números de teléfono destino.

Ejemplo:

- Envía un mensaje a Carmen Martínez que diga Te recuerdo que hemos quedado a las tres de la tarde en la Avenida Libertad.
- Mandar un sms al teléfono 555 996 854 con el texto Tengo una llamada perdida suya, si desea contactar conmigo ya estoy disponible.

III.4.4.1.1.6 Crear una nueva nota

Genera una nueva nota y la almacena en el móvil.

Ejemplos:

- Nota nueva la tapicería del nuevo coche debería ser negra.
- Nota con título Tareas necesarias a realizar esta tarde que dice Es importante que revise las líneas internas de oficina vodafone.

- Crear nota que diga recoger a los niños del colegio.

III.4.4.1.1.7 Añadir una nueva tarea

Añade una nueva tarea en el móvil. Se puede indicar una fecha y una descripción de la tarea.

Ejemplos:

- Tarea nueva el 15 del 7 tengo que ir a Barcelona.
- Inserta tarea el martes día 4 de noviembre tengo una comida con Emilio Iborra en Lorca.
- Tarea en los próximos días tengo que enviar presupuesto a Neosistec.

III.4.4.1.1.8 Establecer la alarma

Establece la alarma del móvil a la hora indicada.

Ejemplos:

- Pon la alarma a las 9.
- Despiértame mañana a las 10 de la mañana.
- Establece a la alarma esta tarde a las 6.

III.4.4.1.1.9 Establecer perfil

Ajusta el perfil de sonido del móvil.

Ejemplos:

- Establece el perfil normal del teléfono.
- Pon el móvil en modo vibración.
- Silenciar el dispositivo.

III.4.4.1.1.10 Consulta historial de llamadas

Previa petición por parte del usuario, la aplicación realiza una lectura de las llamadas perdidas, entrantes o salientes. Finalizada la locución del sistema, se ofrece al usuario la posibilidad de realizar una rellamada a cualquiera de los registros expuestos.

Por ejemplo:

- USUARIO [U]: Dime cuales son las llamadas pedidas del día.

- APLICACIÓN MÓVIL [AM]: Tiene tres llamadas perdidas durante el día de hoy; a las 9:55 le llamó Martín Gómez, a las 10.20 el teléfono 555 896 222 y a las 10:36 Carlos Fernández.

III.4.4.1.1.11 Consulta próximos eventos de la agenda

La aplicación dicta los eventos que el usuario tiene almacenados en su agenda en función de la solicitud realizada por el mismo. Una vez que el usuario dispone de la información solicitada actualizada, se le ofrece la posibilidad de gestionar dicha agenda personal.

Ejemplo:

- [U]: ¿Cuál es mi agenda para la próxima semana?
- [AM]: Tiene tres citas asignadas durante la próxima semana. El lunes 14 a las 17.30 tiene consulta en el médico, el miércoles 16 a las 8.15 viaje a Barcelona, y el jueves 17 a las 9.00 tiene reunión con Abel Ruíz en la Calle Torre de Romo.

III.4.4.1.1.12 Botón de emergencia

La aplicación permite configurar una "lista de emergencia" con números de teléfono listados en orden de prioridad descendente.

Ante una situación de emergencia por parte del usuario, mediante una solicitud verbal para la activación del aviso se producirá una llamada automática al primero de los números almacenados. Si la llamada es atendida se procederá a la activación automática del manos libres del Smartphone; en caso de no haber sido descolgada por el receptor, se llamará al segundo número de la lista, y así sucesivamente.

Ejemplos:

- Realiza llamada de urgencia.
- Aviso de emergencia.
- Llamar urgentemente, necesito ayuda.

Para prevenir que la llamada haya sido atendida por un buzón de voz y el aviso no llegue al receptor con la urgencia requerida, una vez finalizada la misma una locución preguntará al usuario si el aviso se ha ejecutado con éxito, o si desea que se llame al siguiente número de la lista de emergencia. Por ejemplo:

- [AM]: Su llamada de aviso se ha realizado con éxito; ¿desea que se llame al siguiente número de la lista?
- [U]: Sí, llama al siguiente.

III.4.4.1.1.13 Consulta horario

Tras una consulta del usuario, el sistema devuelve la locución con la fecha y hora en curso. Por ejemplo:

- [U]: ¿Qué hora es?
- [AM]: Son las 14.25 del 2 de diciembre de 2.012.

III.4.4.1.2. Funciones activas

III.4.4.1.2.1 Atención llamada entrante.

Se activa una locución automática ante una llamada entrante en curso, notificando al usuario la misma e identificándola con el nombre registrado en la agenda, o con el propio número en caso de no estar registrado.

Una vez realizada la notificación de voz, se le da al usuario las opciones para que a través de voz decida qué acción realizar sobre la misma. Por ejemplo:

- [AM]: Le está llamando Pedro Fernández, ¿que desea hacer: atender la llamada, silenciarla o colgarla?
- [U]: Descuelga.

III.4.4.1.2.2 Notificación de batería baja.

Los terminales convencionalmente emiten un aviso de batería baja mediante notificación en pantalla y un pequeño tono sonoro. La aplicación permite dirimir al usuario a qué se debe dicho mensaje, activando una locución automática notificando que la batería del dispositivo se está acabando, permitiendo además poner el terminal en modo ahorro.

Por ejemplo:

- [AM]: Su nivel de batería es del 10%, su dispositivo se pondrá en modo ahorro de energía.

III.4.4.1.2.3 Sistema Alerta de Accidentes.

Uso del acelerómetro del *smartphone* por parte la aplicación para detectar posibles accidentes domésticos.

Tras diagnosticar una posible caída, el sistema pregunta verbalmente al usuario si se encuentra bien o ha sufrido un percance; a partir de aquí pueden darse dos escenarios:

- Si el usuario contesta y confirma que ha sufrido un accidente y necesita ayuda, la aplicación comienza a realizar el protocolo de avisos a la "lista de emergencia" de manera similar a la explicada en el anterior punto (Emergencia).
- Si el usuario no contesta, el sistema realiza llamadas automáticas a los contactos de la "lista de emergencia", y una locución les notifica que el propietario del teléfono puede estar en una situación de riego. Como medida de seguridad de que el aviso ha llegado al destinatario en la forma adecuada, el sistema da la opción para que el receptor marque la tecla "1" del terminal como confirmación de que ha entendido el mensaje, o la "2" en caso de ser necesario la repetición de la locución.

III.4.4.1.3. Requisitos no funcionales

El sistema deberá ser lo más accesible posible dentro de la funcionalidad indicada, de manera que pueda ser utilizado por el mayor número de usuarios posible.

Por otro lado, los tiempos de respuesta en las operaciones han de ser tales que para el usuario sea transparente la utilización de servicios externos a la Aplicación Móvil para el procesamiento de sus órdenes.

III.4.5. ARQUITECTURA SOFTWARE DE LA APLICACIÓN MÓVIL

En este apartado se describe la arquitectura software desarrollada en los terminales móviles para poder ejecutar los comandos hablados en lenguaje natural. La arquitectura funcional del sistema completo a desarrollar en esta tesis doctoral se mostraba en la Figura 19. En primer lugar, la aplicación cliente, ejecutada en el terminal móvil, es la herramienta de acceso al sistema, su función se basa en enviar el comando de voz del usuario al servidor ASR que se encarga de transformar el comando de voz en texto plano. Una vez que este texto es recibido por el terminal móvil lo reenvía al servidor de procesamiento de comandos que se encarga de analizar este texto mediante herramientas de procesamiento de lenguaje natural y construir un comando

estructurado en XML que describe la acción que ejecutará el terminal cuando reciba el XML. Por último, cuando el terminal móvil recibe el mensaje en formato XML simplemente ejecuta la acción que éste mensaje lleva descrita.

La implementación de la aplicación cliente variará entre las distintas plataformas móviles seleccionadas aunque su arquitectura funcional es la misma en todos los casos y se puede observar en la Figura 51.

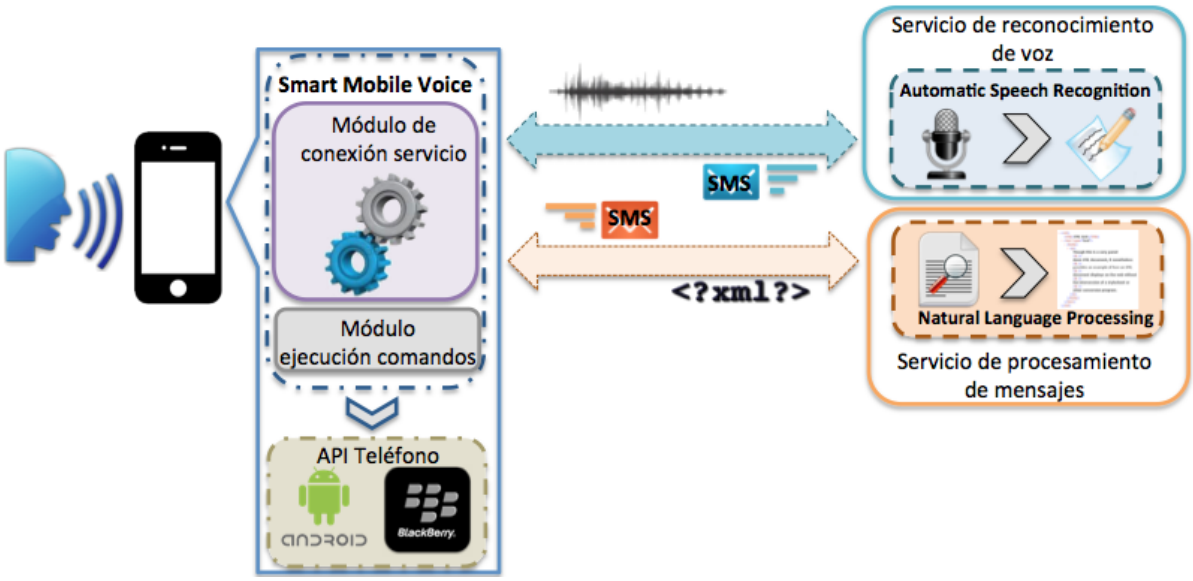


Figura 51. Arquitectura funcional cliente móvil

Como puede apreciarse en la Figura 51, la aplicación se basa principalmente en dos módulos funcionales: El módulo de conexión con el servicio se encargará de obtener el comando estructurado de respuesta a partir del análisis de reconocimiento de voz. El módulo de ejecución de comandos será el encargado de actuar sobre el teléfono para ejecutar el comando recibido. Ambos módulos tendrán una definición idéntica en las plataformas seleccionadas, variando la implementación en lo que se refiere a la interacción con la plataforma móvil.

El resto de la aplicación cliente está compuesta básicamente por la interfaz de usuario, absolutamente dependiente y diferente, incluso a nivel conceptual, en cada una de las plataformas.

Los siguientes apartados describen en detalle cada módulo que constituye la aplicación terminal móvil.

III.4.5.1. Módulo conexión servicio

El módulo de conexión al servicio tiene la importante misión de gestionar las conexiones a los servicios web que se encargarán de, por un lado transformar el comando hablado en texto plano y por otro convertir este texto plano en un comando estructurado basado en tecnología XML que será proporcionado al módulo de ejecución de comandos. La Figura 52 muestra una representación gráfica del funcionamiento del módulo de conexión servicio.

Figura 52. Funcionamiento del módulo de conexión servicio

Como se puede ver en la Figura 52, el funcionamiento del sistema comienza con el dictado del comando, como por ejemplo: “Inserta una reunión a las 9 el próximo martes en el CEEIM con Miguel Ángel”. Después la aplicación terminal compone un mensaje con este audio para enviárselo al Servidor ASR que recibirá el mensaje y lo traducirá en texto enviado de vuelta a la aplicación móvil. Posteriormente, este texto será enviado al servidor que implementa el motor de NLP que recibirá el mensaje y lo transformará en un documento XML que describe el comando a ejecutar.

III.4.5.2. Módulo ejecución comandos

El principal objetivo de este módulo es convertir el documento estructurado en formato XML, en actividades que puedan ser ejecutadas por la plataforma móvil subyacente. La Figura 53 muestra una representación gráfica del funcionamiento del módulo que explicaremos posteriormente.

Figura 53. Funcionamiento del módulo de ejecución de comandos

El módulo de ejecución precisa para funcionar de un comando estructurado en formato XML proporcionado por el módulo de conexión de servicio. Para la ejecución del comando descrito en XML el módulo de ejecución interpreta cada uno de los atributos especificados en comando estructurado y rellena con estos atributos los parámetros necesarios para crear la actividad descrita en el comando. Una vez acabado este proceso de traducción, la actividad se crea dentro de la plataforma y se ejecuta. Este módulo se comunicará con las distintas aplicaciones móviles como agenda, SMS, correo electrónico o teléfono.

III.4.6. DISEÑO DE LA PLATAFORMA

III.4.6.1. Casos de Uso

La Figura 54 presenta a través de un diagrama de caso de uso la descripción de las actividades que se llevan a cabo en el terminal móvil para ejecutar un comando dictado oralmente.

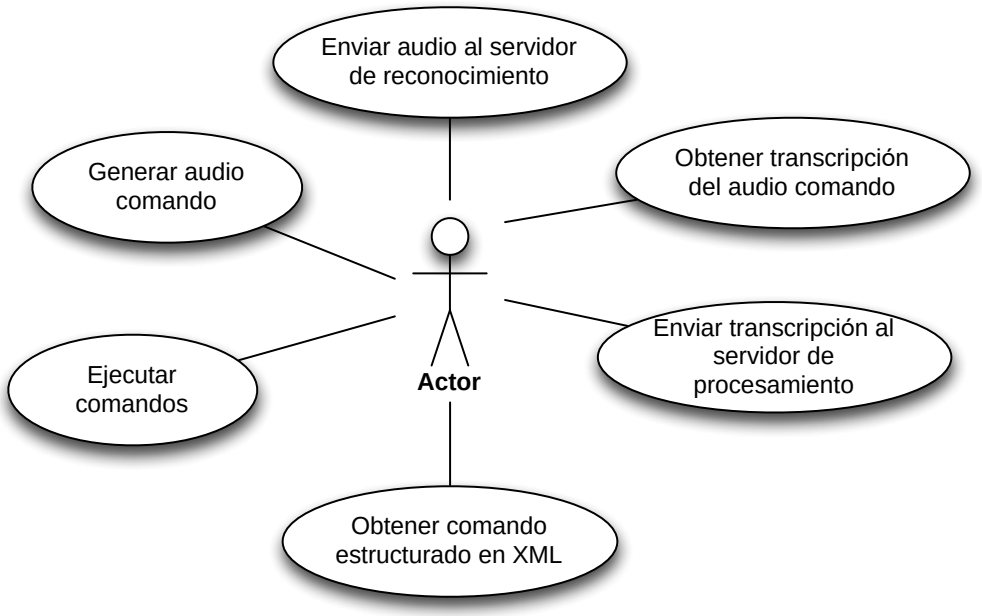


Figura 54. Caso de uso de la aplicación terminal móvil.

III.4.6.2. Configuración / Uso de la aplicación

III.4.6.2.1. Configuración de la aplicación

Una vez descargada la aplicación de la tienda, se procederá a instalar la misma. Finalizado este proceso aparecerá el icono de la aplicación en el menú general del *smartphone*.

Al seleccionar el icono de la aplicación y acceder a la misma, se podrá definir la configuración de la aplicación permitiendo realizar los siguientes ajustes:

- Segundos en pantalla principal: a través de esta opción de configuración se podrá establecer el número de segundos de pausa que realiza la aplicación hasta que indique que se puede decir la instrucción de voz (por defecto son 5 segundos). Aplicable para el inicio de la petición en las funcionalidades tipo bajo demanda.
- Activación automática del altavoz del dispositivo: esta opción permite que la aplicación utilice el manos libres cuando se está usando la aplicación, de forma que se pueda emplear cuando se realicen actividades que impidan usar el móvil correctamente. Por defecto estará activada.

- Lista de emergencia: posibilidad de definir el listado de los cinco números de teléfonos destinatarios sobre los que aplicar el protocolo de emergencia. Estos números podrán introducirse directamente o ser seleccionados desde la agenda del usuario.
- Parámetros del Control de Distancia: definición de la ubicación del punto central (seleccionable mediante mapa o introducción directa de coordenadas) y el radio máximo de control en metros.
- Visualización: Elección por parte del usuario del tipo de visualización, especialmente útil para usuarios con discapacidades visuales específicas.

III.4.6.2.2. Comenzar a usar la aplicación

Para los casos de funcionalidades bajo demanda descritos en el documento de requisitos tras iniciar la aplicación, se accede a la misma, la cual mostrará una pantalla indicando que puede comenzar a dictar la acción que desea realizar. Una vez dictado el comando, el sistema esperará en segundo plano a que se transcriba la orden para enviarla al servidor de análisis semántico.

En el caso de la modalidad activa, la aplicación queda funcionando en segundo plano a la espera de la entrada de una llamada, un aviso de batería baja, una detección de posible caída, o un exceso de distancia respecto al centro de control, activándose de forma automática sin necesidad de intervención por parte del usuario.

III.4.6.2.3. Análisis semántico del comando transcrito.

Recibida la trascripción del mensaje hablado, la aplicación conecta con la plataforma de Procesamiento del Lenguaje Natural enviando la información y esperando la interpretación del mismo.

La aplicación informa en todo momento al usuario de qué está ocurriendo aunque este paso no requiere de interacción por parte del usuario.

III.4.6.2.4. Obtención de la respuesta.

Una vez realizada la interpretación semántica del comando dictado, aparecerá en la pantalla información desglosada de la orden, de forma muy simplificada y visual para el usuario.

III.4.6.2.5. Visualizar el comando ejecutado.

Si el usuario desea consultar el resultado de la acción que ha ejecutado puede pulsar en el botón relacionado que aparece en la pantalla con los resultados en su aplicación, o bien acceder directamente a la aplicación del móvil donde se han registrado los cambios, las cuales en muchos casos se abrirán de forma automática como forma de mostrar al usuario el comando realizado.

III.4.6.3. Interfaz y experiencia de usuario en aplicación Blackberry

Como se ha comentado anteriormente, la aplicación cliente se encarga de realizar las siguientes acciones:

1. Recoge el comando de voz del usuario.
2. Envía el comando vocal a un servicio de transcripción de voz a texto.
3. Recibe la transcripción textual del comando.
4. Una vez obtenido el texto invoca al servicio web del motor de procesamiento del lenguaje natural.
5. Recibe la respuesta XML de este servicio y lleva a cabo la acción equivalente en el dispositivo móvil.

La aplicación se instala en el teléfono y queda asociada a una tecla rápida, de manera que cuando el usuario pulsa esta tecla durante unos segundos la aplicación se abre y queda a la espera de que le digamos un comando, conectando para ello con el servicio de transcripción de voz a texto (ver Figura 55).

Figura 55. Interfaz móvil de la aplicación.

Una vez se ha recibido el texto correspondiente al comando del usuario se envía el texto y el resto de información al servicio que implementa el motor de procesamiento del lenguaje natural que devolverá un documento XML estructurado con la acción que debe realizar el teléfono (ver Figura 56).

Figura 56. Interfaz de respuesta con el comando XML

Una vez obtenida la respuesta XML el teléfono realiza la acción correspondiente y nos muestra el resultado (ver Figura 57).

Figura 57. Acción realizada en el terminal móvil

III.4.6.4. Interfaz y experiencia del usuario en aplicación Android

En este apartado se va analizar la aplicación desde el punto de vista gráfico. En los diferentes apartados que componen este punto se explicaran, a través de capturas de pantalla de la aplicación, los aspectos más relevantes relacionados con la interfaz gráfica y la experiencia del usuario, donde ha sido depositado gran parte del trabajo del desarrollo, para hacer que el uso de todas sus capacidades sea transparente y sencillo para el mismo.

III.4.6.4.1. Interfaz por defecto

La interfaz por defecto se corresponde a la interfaz inicial mostrada por el dispositivo cuando ejecuta la aplicación. Como se puede ver en la Figura 58, se ha diseñado una interfaz simple y muy visual que permita simplificar la curva de aprendizaje en el uso de la aplicación por parte del usuario.

Figura 58. Interfaz Android por defecto

III.4.6.4.2. Interfaz configurable

Debido a la universalidad buscada han sido diseñadas una serie de interfaces configurables por el usuario. Las siguientes figuras muestran una captura de pantalla de una interfaz configurada con iconografía plana y otra de alto contraste para usuarios con deficiencias visuales respectivamente. En ellas podemos observar los estilos gráficos bien diferenciados.

Figura 59. Interfaz con iconografía del plano.

Figura 60. Interfaz en alto contraste.

Capítulo IV. VALIDACIÓN DEL SISTEMA

IV.1. INTRODUCCIÓN

En este documento se detalla la validación del sistema final por usuarios finales interesados en los resultados de la tesis doctoral.

La arquitectura funcional del sistema global (ver Figura 61) es una arquitectura cliente-servidor basada en tecnología de cloud-computing. Los terminales móviles cuentan con una aplicación que será la encargada de captar el sonido/voz y de enviar esa voz a un servidor que se encarga de convertir la voz en texto (reconocimiento automático de voz o ASR) y posteriormente envía ese texto a una aplicación de servidor, donde se encuentran implementadas diversas tecnologías basadas en Web Semántica y Procesamiento de Lenguaje Natural, con el fin de obtener la semántica de la orden y generar un comando estructurado que posteriormente la aplicación instalada en el terminal móvil es capaz de procesar. Esta opción de que tanto el reconocimiento de voz como la interpretación semántica se hagan en servidores y no en el propio terminal, permite que se puedan utilizar grandes librerías (diccionarios, listas de la palabras, analizadores morfo-sintácticos,...) que consumen grandes recursos computacionales todavía no disponibles en la mayoría de móviles o que hace que el consumo de batería se incremente en aquellos smartphones que sí disponen de esos los recursos suficientes, lo que en definitiva dota al sistema de una alta capacidad de poder prestar los nuevos usos buscados en los objetivos del presente trabajo.

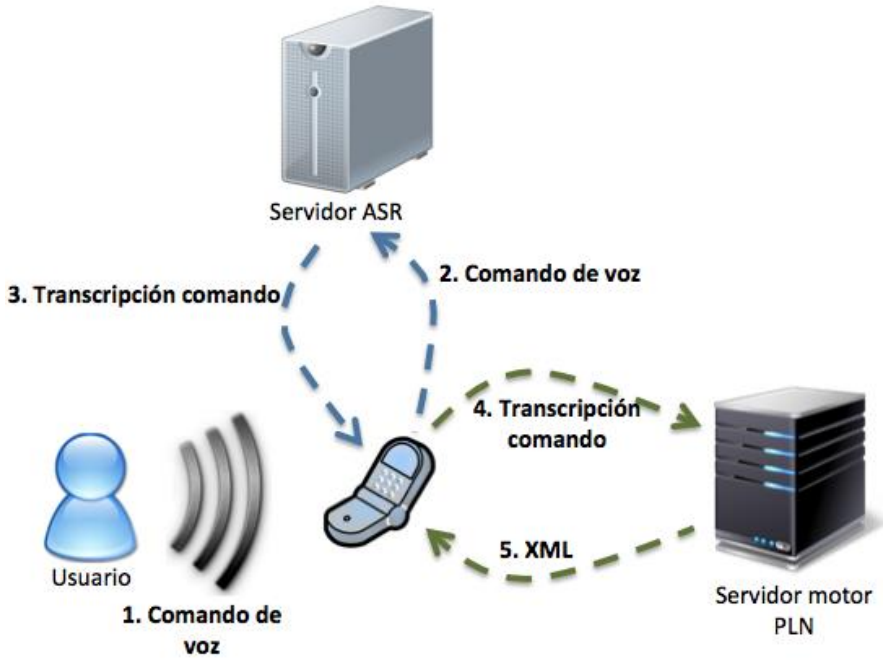


Figura 61. Diseño global del sistema final.

La Figura 61 distingue tres módulos independientes que se integran: el servicio de ASR, el servicio del motor NLP y la aplicación móvil.

Anteriormente ya se había realizado una verificación y validación del servicio del motor de NLP que se encarga de transformar un comando escrito en lenguaje natural en un comando estructurado XML. En primer lugar se probó este servicio mediante invocaciones directas, sin el uso del servicio web. Esto permitía evaluar la precisión en la comprensión de los comandos sin otros factores que pudieran afectar de manera indirecta. Para ello se recopilaron cientos de transcripciones para cada uno de los tipos de comando que se pueden ejecutar (gestionar calendario, añadir contacto, etc.) obtenidos directamente de resultados de transcripción del equipo de desarrollo.

Todas estas transcripciones se agruparon en lotes por tipo de comando y se les asignó el resultado esperado en la interpretación por parte del servicio.

Con estos recursos las pruebas podían ejecutarse de forma automática, obteniendo un porcentaje de aciertos para cada tipo de comando lo que permitía identificar con facilidad los puntos importantes de mejora.

Además, el desarrollo se realizó sobre un entorno de integración continua con CruiseControl (servidor de integración continúa). De esta manera, cada vez que un desarrollador realizaba un cambio en la aplicación, el servidor compilaba la nueva

versión del servicio y ejecutaba la batería de pruebas completa. De esta forma se identificaban rápidamente posibles errores y se medían las mejoras introducidas.

La misma batería de pruebas se utilizó para el acceso a través del servicio web. En esta ocasión se realizaban las peticiones correspondientes al servicio web que simulaban la interacción con el dispositivo móvil. De esta manera se probó tanto la corrección de los resultados como la capacidad de respuesta del servicio web ante situaciones de estrés.

Además, también se realizaron pruebas unitarias en el software del dispositivo móvil, pero no se validó hasta esta tarea los servicios del servidor ASR que se explican en el apartado 2 de este documento.

Una vez verificado y validado el funcionamiento de cada componente, se realizaron las validaciones del sistema global por distintos usuarios. Esta validación se explica en el apartado 3.

IV.2. PRUEBAS EN SERVIDOR ASR

El servidor de ASR requirió un conjunto de pruebas con distintos hablantes para determinar la efectividad del mismo con distintos móviles, micrófonos y personas. Para ello se grabaron distintas pruebas con distintos comandos para manejar las aplicaciones móviles por un conjunto de 10 personas.

Por cada persona se recogieron 30 muestras distintas a 16 kHz con distintos móviles. Entre estas muestras 15 de ellas fueron con comandos predeterminados que los usuarios debían leer de un documento y otras 15 con comandos inventados por el propio usuario. Estos últimos comandos fueron transcritos por los usuarios.

A continuación mostramos algún ejemplo de los 15 comandos predefinidos comunes en todas las muestras:

- “Inserta una alarma a las 9 de la mañana”
- “Añade una tarea para recordarme que tengo que enviar la información a los portugueses”
- “Insertar nota he leído un artículo en la sección de economía de El País que habla que el mercado de aplicaciones móviles va a crecer un 800% hasta 2014”
- “Envía un SMS al teléfono 639011266 que diga por favor confírmame que has recibido el correo electrónico que te envié.”
- “Llamar a Pedro Vivancos”

- "Inserta en el calendario que mañana tenemos una reunión con INDRA a las 10 de la mañana"
- "Crear un nuevo correo electrónico para Javier Pita con asunto pruebas y cuerpo hola Javier, el informe de pruebas te lo enviaremos en los próximos días gracias un saludo."

Para comprobar la efectividad de la transcripción de la voz a texto se comparó el texto transcrito por el ASR y el texto transcrito manualmente para observar qué porcentaje de acierto tenía el sistema. Para ello se utilizó la fórmula mostrada en la Figura 62 basada en la distancia de Levenshtein para calcular el grado de similitud entre dos cadenas de texto. Este algoritmo nos permite obtener la distancia de edición o distancia entre palabras, entendiendo por distancia al número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra. Para normalizar a 1 el resultado hemos diseñado la siguiente fórmula matemática.

$$Similitud\_texto = 1 - \frac{(distancia\ Levenshtein(texto\ comando, texto\ transcrito))}{Máxima\ Longitud(texto\ comando, texto\ transcrito)}$$

Figura 62. Similitud de cadenas de texto basada en la distancia de Levenshtein.

La distancia de Levenshtein es el número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra. Se entiende por operación, bien una inserción, eliminación o la sustitución de un carácter. Por ejemplo, si tenemos las siguientes palabras para comparar: "Elephant" y "Relevant", si aplicamos la definición, la distancia de edición entre ambas palabras aplicando el algoritmo sería de 3 tal y como muestra la Figura 63.

		E	L	E	P	H	A	N	T
	0	1	2	3	4	5	6	7	8
R	1	1	2	3	4	5	6	7	8
E	2	1	2	2	3	4	5	6	7
L	3	2	1	2	3	4	5	6	7
E	4	3	2	1	2	3	4	5	6
V	5	4	3	2	2	3	4	5	6
A	6	5	4	3	3	3	3	4	5
N	7	6	5	4	4	4	4	3	4
T	8	7	6	5	5	5	5	4	3

Figura 63. Ejemplo de cálculo de la distancia de Levenshtein.

Los resultados de esta prueba se pueden consultar en la Figura 64. Como se puede observar la media del reconocimiento de los comandos predeterminados que los usuarios tuvieron en el reconocimiento de los comandos que eran predeterminados fue bastante alta 93,65%. En este reconocimiento el mejor valor lo obtuvo el usuario 9 con un 97,01% de acierto. Por otro lado, la media del reconocimiento de comandos inventados fue del 83,07% debido a que la improvisación en el dictado de comandos propios perjudicó notablemente el reconocimiento debido a las pausas introducidas entre la conversación. Aun así, el peor resultado obtenido fue del 75,87% para el usuario 7 que sería un resultado bastante aceptable. Por último, la media global fue de un 88,63% que representa que el servidor de ASR tiene un muy buen comportamiento con las pruebas de validación realizadas.

	U1	U2	U3	U4	U5	U6	U7	U8	U9	U10	Media
Media comandos predeterminados	94,50%	93,34%	96,24%	94,11%	93,45%	90,02%	89,64%	92,50%	97,01%	95,67%	93,65%
Media comandos inventados	84,58%	82,46%	87,81%	83,86%	82,66%	76,54%	75,87%	80,94%	89,26%	86,74%	83,07%
Media global	89,54%	87,90%	92,02%	88,99%	88,05%	83,28%	82,76%	86,72%	93,13%	91,21%	88,36%

Figura 64. Resultados validación ASR.

IV.3. VALIDACIÓN POR USUARIOS FINALES

Las pruebas de validación del sistema global se componen de dos pruebas conocidas como prueba α y prueba β representadas en la Figura 65. Es importante definir qué pruebas componen esta validación y sobre todo que personas están implicadas en la ejecución de cada estrategia. Más concretamente, esta estrategia se compone a su vez de dos pruebas, la *prueba* α que se lleva a cabo por un cliente en el lugar de desarrollo del software, utilizando al desarrollador principal del software como observador del uso de la aplicación. La función del desarrollador es el registro de errores y problemas de uso que observe durante la prueba de uso del cliente para posteriormente corregirlos, y la *prueba* β que se lleva a cabo por los usuarios finales del software que tienen la

misión de informar de todos los problemas que encuentran durante la prueba beta a los desarrolladores que se encargan de ultimar el software analizando estos informes y corrigiendo los fallos detectados para obtener el sistema final.

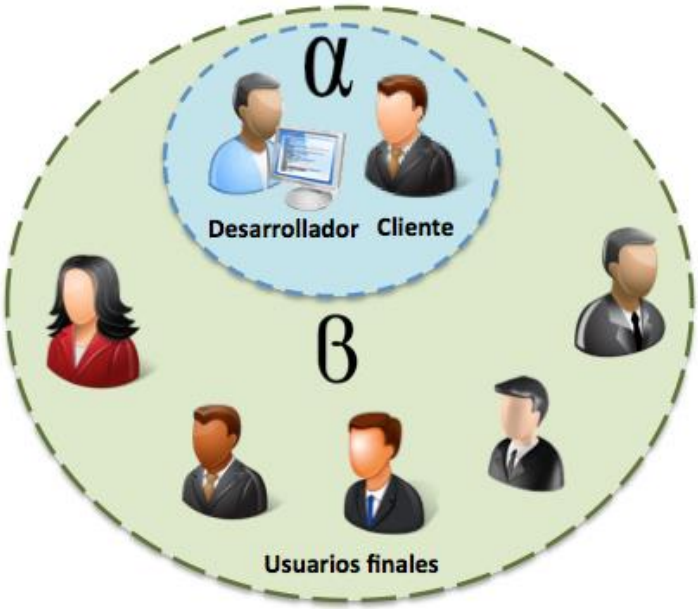


Figura 65. Estrategias de validación del sistema.

Lo primero que se realizó en esta fase fue el diseño de un test sociolingüístico (ver Anexo I) que permitiese detectar sin influir en los usuarios finales, cuál sería la forma natural de interactuar por la voz con un dispositivo móvil. Una vez diseñado este test, se contactó con distintas Asociaciones de Discapacitados como COCEMFE y ONCE y la Fundación para los Estudios de Ingeniería Aplicada a la Integración del Discapacitado. (FEID). Se les presentó el trabajo a cada entidad y se les solicitó que pasaran el test sociolingüístico a distintas personas discapacitadas que estuvieran interesadas en utilizar el sistema.

Se obtuvieron unas 30 respuestas a dicho test con las formas típicas en las que un usuario podría interactuar con las aplicaciones propuestas del dispositivo móvil y otras sugerencias de aplicaciones.

Estas propuestas de comunicación se probaron por el equipo de desarrolladores del trabajo para comprobar la cobertura del mismo con las órdenes que habían propuesto los usuarios dando unos resultados cercanos al 100% de éxito ya que las formas de comunicación usadas por la mayoría de los usuarios eran bastante sencillas y estaban semi-estructuradas.

Una vez realizada esta validación, se seleccionaron a un conjunto de 10 usuarios finales a los que se les dejó un dispositivo móvil durante una semana. Los usuarios podrían entonces utilizar el mismo durante ese tiempo manejando el dispositivo y se monitorizarían todos los resultados para poder determinar la precisión y fiabilidad del sistema.

El resultado de este experimento fue que el uso que le dieron al Smartphone fue bastante pequeño con una media de 5 usos al día por todos los usuarios. En la Figura 66 se muestra el número de comandos por aplicación y por usuario durante todo el periodo. Como se puede observar, las aplicaciones más utilizadas fueron con diferencia el teléfono y SMS con una media de 10,3 y 9,3 usos respectivamente.

	U1	U2	U3	U4	U5	U6	U7	U8	U9	U10	Media
Agenda	5	7	4	3	8	6	2	3	1	1	4
Calendario	7	8	7	6	9	4	3	2	4	3	5,3
Notas	1	2	1	0	0	1	1	1	2	1	1
Tareas	2	1	1	1	0	0	1	2	1	0	0,9
Teléfono	12	10	9	15	10	9	14	9	8	7	10,3
SMS	15	8	7	8	10	8	13	10	8	6	9,3
Email	5	5	8	9	7	3	4	7	3	3	5,4
Total	47	41	37	42	44	31	38	34	27	21	36,2

Figura 66. Resultados validación ASR.

También se les pidió que rellenaran una encuesta de satisfacción sobre la satisfacción del usuario con el sistema y si cambiarían o añadirían alguna funcionalidad.

En general la satisfacción con los resultados obtenidos por la aplicación fue bastante alta y casi todos los usuarios estaban muy satisfechos con los resultados obtenidos, pero cabe destacar que la mayoría de ellos echaron de menos poder interactuar con aplicaciones de mensajería instantánea como WhatsApp o Line.

IV.4. SISTEMA INTEGRADO

El proceso de desarrollo de software no sólo se basa en la generación de código fuente, si no que se encuentra compuesto por diferentes fases que a su vez están compuestas por diferentes actividades. Entre todas estas actividades, la fase de pruebas contempla diferentes tipos de actividades relacionados con la validación y verificación del software. Concretamente, este documento describe las técnicas de integración y

validación del software más utilizadas en los procesos de desarrollo de software. La utilización de estas actividades en la fase de integración facilitó la detección de errores de funcionamiento asociados a las integraciones de los componentes del sistema.

Actualmente, existen diferentes técnicas de pruebas de integración y validación del software. Estas técnicas nos permiten ser más eficientes en la corrección de errores durante los procesos de integración de componentes proporcionando estrategias que facilitan la correcta construcción del sistema de acuerdo a lo especificado en el documento de requisitos.

IV.4.1. TÉCNICAS DE INTEGRACIÓN Y PRUEBAS DE SOFTWARE

La integración final de software es una de las fases más críticas en el desarrollo de software debido al desarrollo en módulos del sistema diseñado. Normalmente, este proceso de integración supone un gran problema para las factorías de software, que además se intensifica cuando los módulos son desarrollados por diferente personal y cuando participan distintas empresas. Aunque a simple vista el proceso de integrar los componentes software dentro de un sistema consiste en la unión de módulos para conseguir que el sistema funcione correctamente de acuerdo a lo especificado, esta tarea puede convertirse en una tarea considerable, dados los errores de interacción que pueden producirse entre los módulos durante su integración. Para que estos problemas no supongan el fracaso del trabajo y un gasto excesivo de recursos debe implementarse una adecuada estrategia de integración que defina los pasos a seguir para facilitar la construcción del sistema final.

En términos generales, la función esencial de las estrategias de integración es definir un orden en el que los módulos tienen que ser integrados para construir el software global. Básicamente, como ya se analizará en detalle posteriormente, existen dos tipos de integraciones intrínsecamente relacionadas con el tipo de software desarrollado, mientras que un tipo se basa en la integración de módulos más generales a más específicos, es decir, de los módulos más cercanos al usuario a los más internos, el otro tipo define justamente el proceso contrario, partir de los módulos más específicos a los más generales. Ambas estrategias se repiten tantas veces como niveles de módulos existan en la aplicación y después de cada proceso de integración se realiza una prueba de regresión. Esta prueba nos permite validar que todo el sistema integrado hasta el

momento funciona correctamente, debido a que cada vez que un módulo es integrado, independientemente de la estrategia de integración seleccionada, el software cambia y estos cambios pueden causar problemas con funciones ya integradas que antes trabajaban correctamente. El hecho de llevar a cabo una prueba de regresión después de cada modificación del software ayuda a asegurar que los cambios introducidos por la integración de nuevos módulos no producen comportamientos no deseados o errores adicionales.

Por otro lado, una vez integrado el sistema final es necesario la realización de un conjunto de pruebas de software. Las pruebas de software son investigaciones empíricas y técnicas cuyo objetivo es proporcionar información objetiva e independiente sobre la calidad del producto a la parte interesada o stakeholder. Es una actividad más en el proceso de control de calidad.

Las pruebas son básicamente un conjunto de actividades dentro del desarrollo de software. Dependiendo del tipo de pruebas, estas actividades podrán implementarse en cualquier momento de dicho proceso de desarrollo. Existen distintos modelos de desarrollo de software, así como modelos de pruebas. A cada uno corresponde un nivel distinto de involucramiento en las actividades de desarrollo.

Las pruebas de software pueden clasificarse en dos tipos:

- Pruebas estáticas: Pruebas que se realizan sin ejecutar el código de la aplicación. Algunos ejemplos de estas pruebas pueden ser: análisis de documentos de requisitos o revisión de documentación asociada al software. Estas pruebas facilitan “pruebas de escritorio” que permiten seguir los flujos de la aplicación.
- Pruebas dinámicas: Pruebas que para aplicarlas necesitan la ejecución del código. Estas pruebas permiten el uso de técnicas de enfoques de pruebas como caja negra que se centran en el análisis de las entadas que recibe cierto módulo y las salidas que produce, sin tener en cuenta los procedimientos internos, y caja blanca, también conocidas como pruebas de caja de cristal o pruebas estructurales, se centran en los detalles procedimentales del software, con mayor amplitud. La ejecución de pruebas dinámicas permite medir con mayor precisión el comportamiento de la aplicación desarrollada.

Además, cada prueba incluye a su vez diferentes tipos de pruebas. La Figura 67 representa las dos agrupaciones de pruebas analizadas con cada uno de los tipos que

incluyen. En concreto, para este sistema, como se desarrollará en los siguientes apartados, se ha seleccionado una prueba dinámica debido a que este tipo de pruebas nos proporciona una visión más específica del comportamiento de la aplicación a través del uso de técnicas que facilitan la detección y corrección de errores de software durante el funcionamiento de la plataforma. Dentro de las pruebas dinámicas, como se puede ver en la Figura 67, existen diferentes técnicas. Cabe destacar que cada una de estas técnicas contienen a su vez conjuntos de métodos diferentes que logran la validación de software de manera distinta. Para el sistema implementado, la técnica dinámica que se ha escogido para realizar los procesos de integración ha sido la técnica basada en niveles de pruebas detallada en la Figura 67.

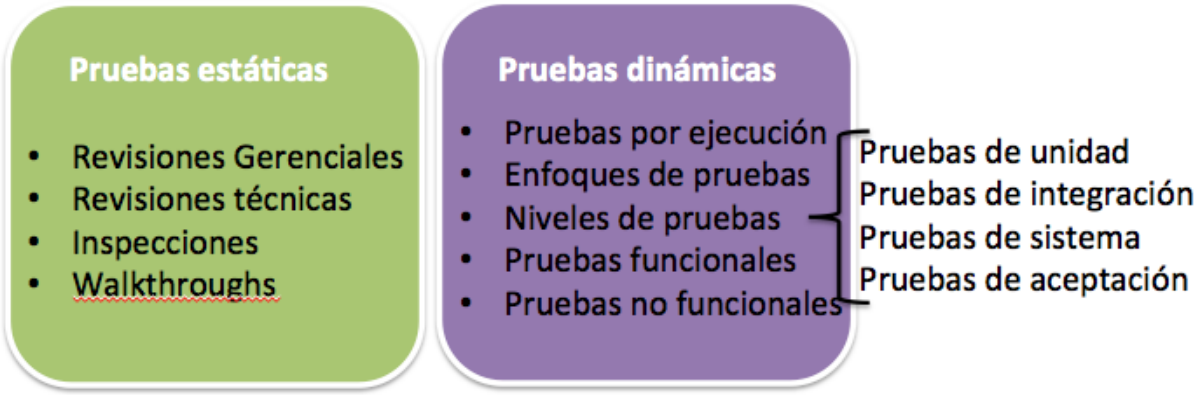


Figura 67. Tipos de pruebas de software

Esta técnica ha sido seleccionada debido a la diversa granularidad que proporciona, como muestra la Figura 67, esta estrategia dinámica se encuentra compuesta a su vez por varios tipos de pruebas que describiremos a continuación:

- Pruebas de unidad: se prueba por separado cada "elemento mínimo de procesamiento" definido por la organización (objetos, componentes, módulos, funciones). Las técnicas más utilizadas son las de caja blanca. Típicamente realizadas por el equipo desarrollador.
- Pruebas de integración: se verifica la interacción entre unidades con técnicas como pruebas de interacción y de mutación. Suelen ser ejecutadas por el equipo de prueba.
- Pruebas de sistema: se verifica el sistema como un todo, aplicando pruebas de caja negra utilizando perfiles de usuario, haciendo pruebas de configuración,

seguridad, confiabilidad y recuperación. Debieran ser ejecutadas por el equipo de prueba.

- Pruebas de aceptación: se verifica la satisfacción de requerimientos con técnicas como pruebas-α y pruebas-β. Debieran ser llevadas a cabo por un equipo que incluya al cliente, usuarios y los desarrolladores expertos.

Como ya ha sido mencionado anteriormente, después de cada iteración de la estrategia definida para la integración del software es necesaria la realización de una prueba de regresión para verificar la consistencia del software con la inclusión de los nuevos módulos. La Figura 68 muestra los diferentes tipos de pruebas de regresión existentes.

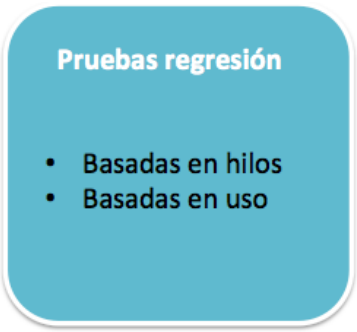


Figura 68. Tipos de pruebas de regresión.

La Figura 68 muestra los distintos tipos de regresión más extendidos en la industria del software. La diferencia existente entre ambas es que mientras las pruebas de regresión basadas en hilos requieren la implementación de una serie de clases que responden a una entrada o evento del sistema, las pruebas basadas en uso comienzan con la comprobación de las clases más externas del sistema (clases independientes) y poco a poco se va interiorizando y comprobando las clases internas (clases dependientes). En concreto, para el sistema implementado la prueba de regresión seleccionada para utilizarse durante el proceso de integración ha sido la prueba basada en uso. Esta es quizás una prueba más pesada y trabajada desde el punto de vista de la comprobación cuando se produzca un error, ya que se deben comprobar los diferentes niveles de clases implementadas que irán aumentando de manera directamente proporcional al número de módulos a integrar. La parte positiva es que genera un código más limpio ya que no requiere de la implementación de nuevas clases que respondan ante una entrada o evento del sistema.

Una vez que las pruebas de integración han sido satisfactorias, el siguiente paso es llevar a cabo las pruebas de validación. En la ingeniería de software las pruebas de validación son el proceso de revisión que verifica que el sistema de software producido cumple con las especificaciones y que logra su cometido. Es normalmente una de las partes finales del proceso de pruebas de software de un proyecto, que también utiliza técnicas tales como evaluaciones, inspecciones y tutoriales. La validación es el proceso de comprobar que lo que se ha especificado es lo que el usuario realmente quería. Existen diferentes pruebas de validación y tal y como se muestra en la Figura 69.

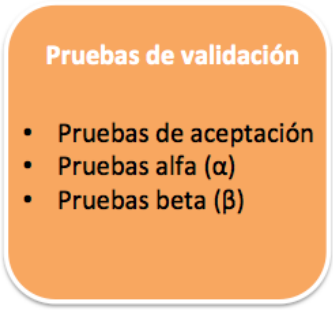


Figura 69. Tipos de pruebas de validación.

A modo de introducción, la diferencia entre las pruebas de validación representadas en la Figura 69 es que las pruebas de aceptación están desarrolladas por los clientes, las *pruebas α* son llevadas a cabo por un solo cliente teniendo al desarrollador como observador, y por último las *pruebas ß* son aquellas pruebas de validación que se realizan por los clientes en el entorno de trabajo y sin ningún observador. Las pruebas de validación que se han implementado en este sistema son las *pruebas α* y *las pruebas ß* que se describen en el apartado siguiente donde se especifica como han sido aplicadas ambas pruebas.

IV.4.2. Diseño y definición de la integración y validación de los componentes.

En primer lugar, antes de entrar en detalle en el diseño y definición de la integración y la verificación de los componentes, se proporciona la Figura 70 para recordar la arquitectura diseñada del sistema global.

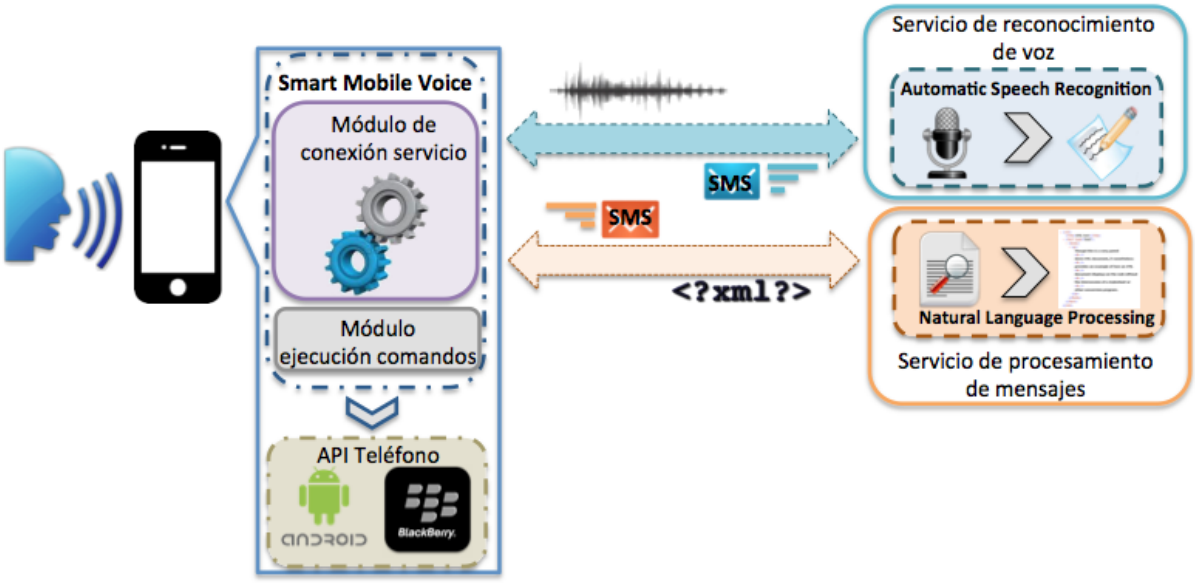


Figura 70. Descomposición en componentes del sistema

Como muestra la Figura 70, la arquitectura funcional del sistema planteado es una arquitectura cliente-servidor. Los terminales móviles cuentan con una aplicación que será la encargada de captar el sonido/voz y de enviar esa voz a un servidor que se encarga de convertir la voz en texto (reconocimiento automático de voz o ASR) y posteriormente envía ese texto a una aplicación de servidor, donde se encuentran implementadas diversas tecnologías basadas en Web Semántica y Procesamiento de Lenguaje Natural, con el fin de obtener la semántica de la orden y generar un comando estructurado que posteriormente la aplicación instalada en el terminal móvil es capaz de procesar. Esta opción de que tanto el reconocimiento de voz como la interpretación semántica se hagan en servidores y no en el propio terminal, permite que se puedan utilizar grandes librerías (diccionarios, listas de la palabras, analizadores morfo-sintácticos,...) que consumen grandes recursos computacionales todavía no disponibles en la mayoría de móviles o que hace que el consumo de batería se incremente en aquellos smartphones que sí disponen de esos los recursos suficientes, lo que en definitiva dota al sistema de una alta capacidad de poder prestar los nuevos usos buscados en los objetivos de la presente tesis doctoral. Además, esta arquitectura está en harmonía con los fines comerciales previstos como resultado de este trabajo y que consisten en que las operadoras puedan ofrecer esta tecnología como un servicio a sus clientes.

La arquitectura planteada permite al usuario, simplemente pulsando un botón de su terminal móvil, iniciar la aplicación que permitirá captar la voz del usuario para que ésta sea transcrita usando un servicio instalado en un servidor ASR para la conversión de voz a texto. Una vez realizada dicha conversión, el texto reconocido será enviado al servicio proporcionado por la aplicación del servidor con el fin de analizar el significado de ese texto con el comando expresado en lenguaje natural para crear un comando estructurado XML que será fácilmente procesable por la aplicación instalada en el terminal móvil, realizando la acción deseada por el usuario. Veamos un ejemplo que ayude a esclarecer cualquier duda que pueda surgir sobre la aplicación y la arquitectura:

1. El usuario pulsa un botón o tecla de acceso rápido que ejecuta el cliente instalado en el smartphone y a continuación dice un comando verbal (p.e. “apúntame una reunión el próximo lunes con Pedro García” [voz]).
2. La aplicación instalada en el móvil y resultado de este trabajo enviará el audio al servidor ASR, capaz de transcribir la voz en texto.
3. Una vez procesado este texto, el servidor ASR enviará a la aplicación móvil el texto transcrito (siguiendo el ejemplo: “apúntame una reunión el próximo lunes con Pedro García” [texto]).
4. Ese texto será ahora enviado por la aplicación del móvil al servicio proporcionado por la aplicación Servidor, resultado de este trabajo, ofrecido como servicio HTTP a través de la red.
5. La aplicación del servidor se encargará de procesar el comando y obtener su significado utilizando las tecnologías que se pretenden desarrollar en esta tesis doctoral.
6. Una vez identificado ese comando la aplicación del servidor creará un comando estructurado en XML con la información legible para la aplicación móvil (en el ejemplo:

```
<xml...>
    <AddCalendar...>
        <startDay>31/10/2011</startDay>
        <guest>Pedro García</guest>
    </AddCalendar>
</xml>
```

Figura 71. Fragmento del comando AddCalendar

7. Una vez recibido el comando estructurado, la aplicación del móvil sabrá qué cambios tiene que reflejar en el móvil, creando un nuevo evento en la agenda del mismo para el día 31/10/2010 e indicando que la reunión se celebrará con Pedro García.

Partiendo de que todas las pruebas unitarias han sido satisfactorias, es decir, que el funcionamiento aislado de cada uno de los componentes ha sido chequeado y es correcto, se llevó a cabo la integración de todos los componentes en la plataforma global. Existen diferentes técnicas de llevar esta tarea a cabo, según el tipo de dependencia funcional existente en los componentes del sistema.

En nuestro caso se ha seleccionado una técnica de integración ascendente. Esta técnica de integración comienza por los módulos más bajos de la arquitectura del sistema hasta los módulos superiores cercanos al usuario. La estrategia de integración que diseñamos para el sistema global fue la siguiente:

- Combinar el módulo del nivel más bajo con el módulo del siguiente nivel.
- Escribir un programa controlador de prueba que permita coordinar las entradas y salidas de casos de prueba programados.
- Probar el grupo de módulos.
- Si la prueba ha sido satisfactoria, se eliminan los controladores y los casos de prueba, se combinan ambos módulos y se da por integrada esta parte de la arquitectura.
- Se aplican pruebas de regresión que permiten chequear todo el sistema integrado evitando comportamientos no deseados o errores adicionales provocados por la integración de nuevos módulos.
- Se pasa al nivel superior donde se repetirá esta estrategia tantas veces como niveles de módulos existan en la arquitectura.

Una vez definida la estrategia de integración ascendente, el siguiente paso fue aplicarla sobre el software desarrollado. La aplicación comenzó en el orden que se dispone en la Figura 72. En los siguientes puntos describiremos como se llevó a cabo esta estrategia.

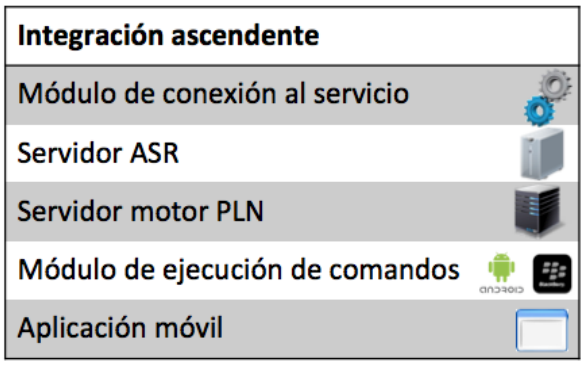


Figura 72. Orden del proceso de integración en el sistema.

1. La estrategia de integración comenzó a aplicarse con el módulo de conexión de servicio junto con el módulo de reconocimiento de voz ubicado en el servidor ASR. Para ello, fueron implementadas numerosos controladores que permitían verificar la coordinación entre ambos dispositivos y comprobar el funcionamiento del reconocedor de voz.
2. Después de validar esta integración, la estrategia de integración se repitió para el módulo que se encuentra en el nivel superior, servidor motor NLP, para la que también fueron implementados diferentes tipos de controladores que se encargaron de comprobar la coordinación entre el módulo de conexión al servicio con el servidor motor NLP en particular. Además también se desarrollaron controladores que permitían comprobar la integración del módulo de conexión con los dos servidores externos.
3. El siguiente paso fue la integración del módulo de conexión al servicio con los servicios de reconocimiento de voz y de procesamiento del lenguaje natural y con el módulo de ejecución de comandos.
4. Por último, una vez que las pruebas de integración de toda la arquitectura software fueron satisfactorias, la siguiente actividad de integración se llevó a cabo entre el sistema integrado con la aplicación móvil. Para este caso las pruebas de integración solo consistieron en la implementación de interfaces gráficas que facilitaran el uso de la plataforma inteligente.

Una vez que la actividad de integración se ha realizado con éxito, la siguiente actividad que se va a llevar a cabo es la prueba de validación. Esta prueba permite saber si el software funciona de acuerdo con la especificación de requisitos definida. Esta prueba compone a su vez de dos pruebas conocidas como prueba α y prueba β representadas

en la Figura 65, donde se presenta la estrategia que se ha utilizado durnate las pruebas de validación, en ella se puede ver representada las dos estrategias que componen esta prueba de validación y sobre todo que personas están implicadas en la ejecución de cada estrategia. Como se describió anteriormente, esta estrategia se compone a su vez de dos pruebas, la *prueba α* que se lleva a cabo por un cliente en el lugar de desarrollo del software, utilizando al desarrollador principal del software como observador del uso de la aplicación. La función del desarrollador es el registro de errores y problemas de uso que observe durante la prueba de uso del cliente para posteriormente corregirlos, y la *prueba β* que se lleva a cabo por los usuarios finales del software que tienen la misión de informar de todos los problemas que encuentran durante la prueba beta a los desarrolladores que se encargan de ultimar el software analizando estos informes y corrigiendo los fallos detectados para obtener el sistema final.

IV.4.3. Implementación de la integración de los componentes.

En este apartado se va a describir el proceso de integración llevado a cabo para cada uno los componentes del sistema. La Figura 61 representa la arquitectura final del sistema integrado, en ella se pueden identificar los diferentes componentes del sistema.

La implementación de la integración de cada componente de la arquitectura se realizó de la siguiente manera:

1. Dispositivo móvil integración con servidor ASR: Después de realizar las correspondientes pruebas unitarias de cada componente por separado se inició la integración del dispositivo móvil (módulo de conexiones a servicios) y el servidor ASR. Para llevar a cabo este proceso de integración previamente tuvo que seleccionarse una forma de comunicación entre ambos dispositivos. La técnica de comunicación seleccionada, como se puede observar en la Figura 73, fue a través de un protocolo seguro de comunicación TSL que proporciona los siguientes beneficios a la infraestructura de red necesaria para desarrollar la función de la aplicación:

    a. Mayor protección – La configuración de TLS en los servidores que constituyen la infraestructura asegura que toda la información que envíen será securizada y enviada a una dispositivo de confianza.

b. Disponibilidad – TLS está disponible en la mayoría de los servidores y es una solución de seguridad aceptada en todo el mundo lo que facilita la integración de la arquitectura en cualquier escenario.

c. Reducción de costos de implementación– La configuración de TLS en los servidores solo requiere la adquisición del certificado digital TLS, a diferencia de muchas otras configuraciones que requieren licencias corporativas.

d. Rápida implementación – La implementación de TLS en los servidores se configura directamente, el proceso de activación es sencillo y no requiere ningún tipo de configuración adicional.

Una vez seleccionada y configurada el protocolo de comunicaciones entre ambas entidades, el proceso de integración continuó con el desarrollo de controladores que verificaran el correcto funcionamiento del módulo de conexiones a servicios, del protocolo de comunicación seguro y del funcionamiento del servidor ASR.

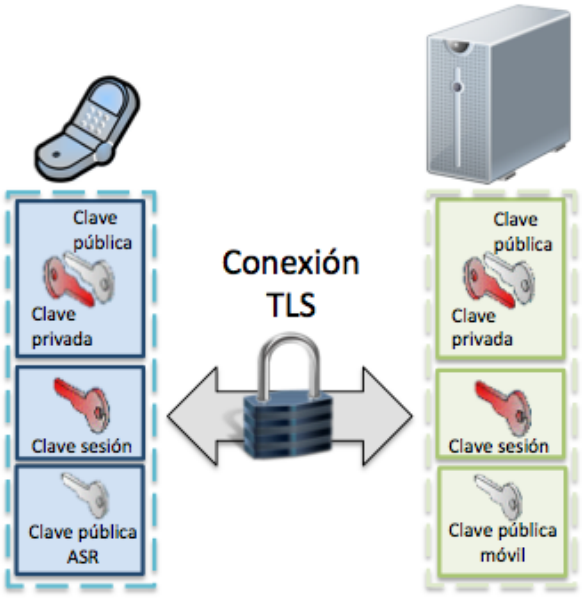


Figura 73. Conexión segura entre dispositivo móvil y servidor ASR.

2. Dispositivo móvil integración con servidor NLP: En este caso, el proceso de integración entre el dispositivo móvil y el servidor NLP, como muestra la Figura 73, se asemeja bastante al explicado en el apartado anterior. La única diferencia

existente es el tipo de servicio al que el móvil accede. Como en el anterior caso analizado, después del despliegue de claves públicas y privadas entre las entidades a integrar y la configuración del protocolo de comunicación TLS, los siguientes pasos del proceso de integración se basan en la comprobación del funcionamiento del módulo de conexiones a servicios, del protocolo seguro de comunicación y del funcionamiento del servidor motor NLP.

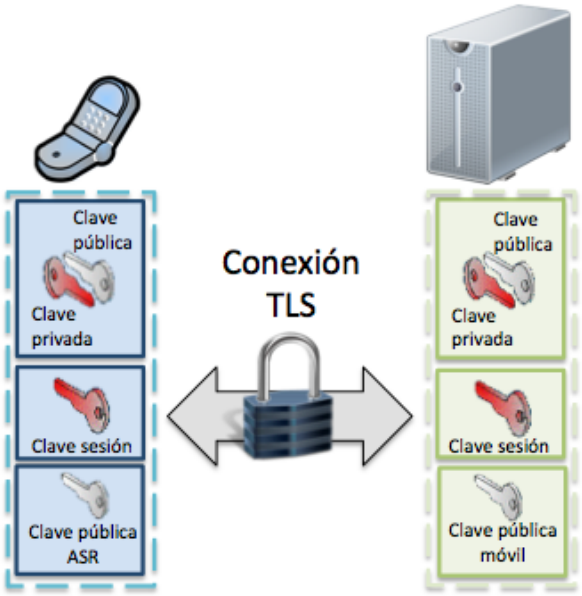


Figura 74. Conexión segura entre dispositivo móvil y servidor ASR.

3. Interfaces de comunicación para crear comando ejecutable: Llegados a este punto del proceso de integración, todos los servicios externos al dispositivo móvil quedarían integrados. Por lo que las siguientes actividades de integración estarían ubicadas en el dispositivo móvil, como en este caso, donde analizamos la integración entre el módulo de conexión al servicio y el módulo encargado de crear el comando que será ejecutado en la plataforma móvil. La primera actividad de integración que se llevó a cabo fue el análisis de la interfaz de comunicación entre ambos módulos, para realizar este análisis fueron implementados varios programas de control de prueba que facilitaron el análisis de las entradas y salidas de la interfaz de comunicación.

4. Aplicación integradora de la plataforma móvil: Ultima actividad de integración que consistió en dotar a la arquitectura propuesta de una interfaz gráfica sencilla dada la universalidad de clientes a la que va dirigida el uso de la aplicación.

Una vez que los procesos de integración y las pruebas asociadas a cada uno de ellos han sido satisfactorias, el siguiente paso fue la validación del software a través de las pruebas α y β que han sido analizadas en el apartado anterior.

IV.4.4. PRUEBAS DE RENDIMIENTO EN LA NUBE

La nube posibilita actualmente dos puntos de vista con relación a las pruebas de software:

- Pruebas del Software en la Nube (Testing Cloud), dirigidas a validar las aplicaciones e infraestructura que están desplegadas en el ecosistema de la nube bajo demanda (on demand), verificando aspectos tales como: disponibilidad, seguridad, interoperabilidad, carga de transacciones, entre otras.
- Pruebas utilizando la infraestructura de la Nube (Cloud Testing), las cuales permiten utilizar el poder de cómputo de la Nube para realizar actividades de validación, utilizando la infraestructura como servicio (IaaS).

La elección de uno u otro enfoque de pruebas en la nube, implica conocer aspectos en común, como lo son los tipos de pruebas en la nube, donde se pueden destacar las siguientes de interés para esta tesis doctoral:

- Pruebas de disponibilidad, orientadas a garantizar la disponibilidad 7x24.
- Pruebas de seguridad, enfocadas a garantizar la integridad de los datos y el acceso seguro a la información.
- Pruebas de rendimiento, que incluyen las de carga (incrementando la demanda de los servicios de la nube vs el tiempo de respuesta) y las de stress (incrementando gradualmente las peticiones para encontrar el punto límite de la nube).
- Pruebas de latencia, buscan determinar fallas en el proceso de envio y recepción de las peticiones en la nube.
- Pruebas de escalabilidad, orientadas a determinar las capacidades de crecimiento de las aplicaciones e infraestructura de la nube.

- Pruebas de resistencia, orientadas a evaluar las incidencias que se pueden presentar en la nube en un tiempo determinado con una carga de transacciones establecida.

Para realizar estas pruebas de rendimiento se utilizó JMeter, que es un proyecto Apache que el proveedor de servicios PaaS tiene incluida en su sistema. Esta herramienta de código abierto muy completa, está implementada en Java y permite realizar tests de comportamiento funcional y de estrés para posteriormente medir el rendimiento. El funcionamiento de esta herramienta se basa en: configurar planes de pruebas y bancos de trabajo. En particular, este documento se centrará en al análisis de los planes de prueba debido al objetivo de análisis rendimiento del sistema diseñado.

Los planes de prueba en JMeter representan conjuntos de elementos que configuran el comportamiento de los test que se llevan a cabo sobre el software. Los elementos que se incorporan en estos planes de pruebas son:

- Thread groups: Representa el punto de partida de cualquier plan de pruebas. Como su nombre indica define el número de hilos que JMeter ejecutará cuando se lancen las pruebas. Cada hilo ejecuta el plan de test de manera independiente a los demás hilos.
- Controladores lógicos: La herramienta define dos tipos de controladores asociados a un ThreadGroup previamente creado:
- Muestreadores / Samplers: Informan a JMeter de cuales son las peticiones que debe enviar.
- Controladores lógicos: Permiten definir cuando se deben enviar las peticiones.
- Sample generating controllers: Indican a la herramienta que realice las emisiones al servidor y espere las respuestas, cada sample define las características de cada petición. En concreto, para el sistema desarrollado el estudio se centrará en aspectos relacionados con las peticiones a servidores web.
- Listeners: Utilizados para monitorizar los resultados asociados a las pruebas, son procesados al final del alcance donde han sido ubicados.
- Timers: Sirven para añadir retrasos entre las distintas peticiones enviadas, debido a que por defecto los threads de los thread groups envían peticiones sin ningún retardo entre cada petición evitando de esta forma la sobrecarga del servidor.

- Aserciones: Se utilizan para realizar comprobaciones adicionales sobre los samplers, existen diferentes tipos: de respuesta, duración, tamaño, etc.

Para cada prueba que ha sido descrita al principio del apartado se definieron diferentes planes de prueba que contenían grupos de hilos thread groups que permitían analizar el rendimiento del sistema. Además, también fueron instalados listeners que ofrecían de manera más representativa el rendimiento del sistema. Por último, fueron configurados varios samplers que permitieron analizar el rendimiento de las peticiones realizadas al servidor, para esta última prueba se incluyeron algunas aserciones que facilitaron comprobaciones adicionales relacionadas con tiempo de retardo, tiempo de acceso, tiempo de extracción de la información de la petición, entre otras.

IV.5. CONCLUSIONES

Este documento recoge la validación definitiva del sistema. Se ha mostrado durante el documento que las pruebas realizadas son muy satisfactorias con unos ratios de funcionamiento muy buenos. Por un lado, los resultados de la transcripción de comandos de voz a comandos de texto son mayores del 88% de precisión. Por otro lado, la validación por parte de los usuarios finales y la experiencia de usuario fue bastante buena aunque el conjunto de personas que validaron el sistema no es muy representativo. Aquí se realizó primero un test sociolingüístico para detectar la manera que tendrían los usuarios de interactuar con el sistema. Seguidamente, se seleccionó un conjunto reducido de usuarios para que probaran la aplicación y dieran sus opiniones acerca de la misma.

Para el futuro se pretende realizar una validación más extensa con muchos más usuarios finales que puedan dar retroalimentación a las interfaces que proporciona el sistema completo.

Capítulo V. CONCLUSIONES Y LÍNEAS FUTURAS

V.1. CONCLUSIONES

Este trabajo comenzó a realizarse a lo largo del año 2010. Por aquellos entonces los *smartphones* de Apple y Google estaban comenzando a penetrar en el mercado y el líder indiscutible de este tipo de dispositivos eran las Blackberry.

Todo esto cambió significativamente con el lanzamiento de Apple Siri, a principios de 2013 (aunque fue anunciado en septiembre de 2012). Esto significó el comienzo del reconocimiento de voz en los dispositivos móviles de Apple y en muy poco tiempo Goole hizo lo propio incluyendo Google Now en su sistema Android.

Para entonces, Blackberry ya apenas podía competir con los dispositivos Android e iOS, y Microsoft se preparaba para lanzar su Windows Phone que también incluía reconocimiento de voz y su asistente de voz llamado Cortana.

Lógicamente, muchas de las decisiones tomadas en esta tesis doctoral se realizaron con un estado de la tecnología disponible que nada tiene que ver con la fotografía actual.

Actualmente cualquier dispositivo móvil de Microsoft, Apple o Android, que superan el 90% del mercado de los *smartphone* en la actualidad, incorpora reconocimiento de voz en tiempo real, por lo que una línea de trabajo clara está en emplear el reconocimiento de voz que incorporan en vez de emplear el servicio DictaSMS de Vodafone. Además, esto permitirá que la aplicación sea válida para cualquier otro operador por lo que se incrementaría el número de posibles usuarios.

Más aún, el reconocimiento de voz que ofrecen los *smartphones* actualmente se realiza en la nube, por lo que no es posible usarlo cuando no se tiene conexión. Sin embargo, el hardware de los *smartphone* ha mejorado increíblemente en los últimos años, y en la actualidad la mayoría de los dispositivos móviles tiene capacidad para realizar el reconocimiento de voz en el propio dispositivo, por lo que una opción sería usar una solución como CMU Sphinx, o una solución comercial como INVOX (www.vocali.net) para este problema.

Por otro lado, las soluciones de Google Now o Apple Siri no sólo han supuesto que los *smartphones* incorporen reconocimiento de voz, sino que son completos asistentes que incluyen muchas de las funcionalidades descritas en este trabajo y otras muchas más, por lo que el resultado de la tesis doctoral se ha quedado obsoleto desde el punto de vista comercial. Sin embargo, se ha creado una interesante tecnología, muy modular, que puede ser fácilmente ampliada con nuevos comandos, y eso significa que es una tecnología que sirve de base no sólo para desarrollar asistentes específicos para teléfonos móviles, sino también otro tipo de servicios inteligentes como sistemas IVR para *call centers* que sean más "inteligentes".

Esta tesis por tanto ha realizado diferentes aportaciones en forma de herramientas para la creación de diferentes asistentes "virtuales" o "inteligentes":

- **Modelo ontológico para la representación del conocimiento del asistente**. Se ha desarrollado un modelo ontológico que permita representar comandos en lengujane natural, esto es que obtenga toda la información semántica de un comando. El modelo ontológico u ontología que representa el conocimiento incluido en un comando en lenguaje natural es un fichero OWL que representa el comando realizada con toda la información semántica que ha sido posible extraer de ella. La finalidad es describir el comando de manera que otros módulos puedan utilizar la información contenida en él para poder interpretarlo y ejecutarlo. Esta ontología se compone de 3 clases principales: 1) aplicación: representa la aplicación software o programa sobre la cual va a recaer la acción. Este puede ser la agenda, alarma, calendario, correo electrónico o SMS; 2) acción: representa la acción a realizar que puede ser insertar, modificar o eliminar; 3) elemento: representa entidades que pueden formar parte de un comando determinado como personas, email, números de teléfono, direcciones, fechas, horas, etc. Para obtener el modelo del comando se han desarrollado diversos recursos lingüísticos que permitan detectar acciones, dispositivos o aplicaciones donde realizar esas acciones y la detección de entidades nombradas como son nombres propios, lugares, direcciones, teléfonos, direcciones de correo electrónico, etc. Dichos recursos se han implementado utilizando la herramienta para el desarrollo de aplicaciones de minería de texto GATE y más

concretamente de sus Gazzeteers y reglas JAPE para poder localizar estas entidades.

- **Motor de representación y análisis de expresiones temporales**. Se ha diseñado un motor para reconocer expresiones temporales utilizando un estándar como TIMEX2, que es una especificación que fue desarrollada inicialmente en el año 2001 en el programa de DARPA's Translingual Information Detection, Extraction and Summarization (TIDES). Las características principales que se añadieron con respecto a TIMEX fue la inclusión de anotaciones para interpretar fechas y horas usando la norma ISO 8601 como el estándar de representación de fechas y horas. La especificación TIMEX2 ha sido usada en distintos proyectos de investigación donde se necesitaba la identificación de expresiones temporales. Este motor de expresiones temporales permite la identificación de tanto expresiones temporales concretas como puede ser "El 5 de marzo de 2011" o bien expresiones temporales más indeterminadas como "mañana a las 4 de la tarde", "el jueves que viene a las 2", "el pasado martes", etc.
- **Recursos lingüísticos para el procesamiento de órdenes vocales relacionadas con el control de aplicaciones del móvil**. Se han creado distintos recursos lingüísticos como gazzetters y reglas JAPE para detectar las aplicaciones a ejecutar en el móvil, así como las posibles acciones a realizar sobre estas aplicaciones. Gracias a estos recursos se pueden detectar los comandos que se pueden realizar sobre las aplicaciones del móvil. Los gazetters representan lexicones o listas de términos de un determinado dominio, y permiten detectar entidades nombradas de personas, lugares, empresas y otros elementos que sean necesarios identificar en las órdenes verbales como emails, urls, direcciones, o teléfonos. Gracias a estos recursos se pueden detectar los trozos de texto que representan entidades nombradas en las órdenes vocales. Más concretamente se han desarrollado distintas listas y lexicones con información sobre personas, empresas y países y otra información relevante en el dominio.
- **Detección y resolución de ambigüedades**. Se realizó un estudio de las distintas aproximaciones de desambiguación léxica actuales. Una forma muy extendida de clasificar los sistemas de desambiguación se basa en la principal fuente de conocimiento utilizada para establecer los distintos sentidos que puede tomar

dicho término. Primero, existen los métodos que utilizan diccionarios, tesauros, bases de conocimiento léxicas como ontologías en lenguaje natural, y reglas de desambiguación sin utilizar ningún corpus. Este tipo de métodos se denominan basados en diccionarios o basados en conocimiento. Por otro lado, existen métodos que evitan tener información externa y trabajan con corpus sin etiquetar. Éstos se suelen denominar métodos no supervisados. Por último, existen los sistemas supervisados y semi-supervisados, que utilizan corpus anotados semánticamente como entrenamiento, o como fuente de datos en un sistema. En este trabajo se decidió desarrollar un sistema basado en conocimiento, reglas y heurísticas para la desambiguación usando la infraestructura GATE y en particular usando reglas JAPE que son apropiadas para detectar, y posteriormente anotar, las expresiones ambiguas y en base al contexto poder decidir qué sentido será el correcto en el comando del dispositivo móvil.

- ***Pipeline* de procesos NLP para procesamiento de comandos**. Para el correcto desarrollo de este trabajo se estudiaron las distintas fases de procesamiento del lenguaje natural para poder unirlas en una aplicación. Con esto, se desarrolló el componente principal del sistema que permitiera procesar los comandos de texto en lenguaje natural dictados por el usuario para la obtención de comandos estructurados que permitan registrar la operación del usuario en el terminal móvil. Para ello se han empleado las tecnologías que se describieron en el documento y el framework GATE para desarrollo de aplicaciones de procesamiento del lenguaje natural. Además, esta solución se basa en la utilización e integración de los recursos lingüísticos desarrollados y los módulos de reconocimiento de expresiones temporales y del sistema de inferencia para la resolución de ambigüedades. En primer lugar se realizó un estudio de diversas tecnologías de ingeniería lingüística para determinar las herramientas de Procesamiento del Lenguaje Natural necesarias para el correcto procesamiento y comprensión semántica de la orden verbal. En esta evaluación se decidió utilizar la herramienta GATE sobre la cual se desarrolló una aplicación compuesta por distintos módulos. La entrada a esta aplicación es un comando escrito en texto en lenguaje natural y la salida es un comando estructurado en el formato

XML y que después se procesará en la aplicación cliente del móvil. Este *pipeline* se basa en cuatro módulos principales: preprocesamiento del texto, detección y resolución de expresiones temporales, detección y resolución de ambigüedades y selección del mejor comando y creación del comando en XML.

- **Generación de comandos estructurados**. Este módulo de transformar las anotaciones no ambiguas obtenidas por el módulo NLP en un comando estructurado XML según el XML Schema descrito en el este documento. Este módulo primero debía comprobar si a partir de las anotaciones obtenidas se puede crear un comando. Para desarrollar esta comprobación se ha desarrollado un conjunto de heurísticas que indican qué elementos son necesarios para poder formar dicho comando. Por ejemplo, un comando para escribir un email deberá tener identificada una anotación Action "Crear", una anotación Application "email" y una anotación que contenga el destinatario del correo electrónico SendTo. Otro ejemplo sería que para poder insertar un contacto en la agenda se debe haber identificado como Application “agenda” la acción (anotación Action) debería ser “insertar” y al menos debería aparecer una Persona. El generador XML se ha implementado utilizando JAXB. Esta librería permite, a partir de un esquema XML generar un conjunto de clases Java que se corresponden con el mismo. Además, automatiza la serialización de estos objetos de acuerdo al propio esquema. Se ha utilizado JAXB para generar las clases a partir del esquema (se deberá hacer en tiempo de compilación). Estas clases son las que el motor PLN devolverá como salida de su procesamiento y que el generador XML transformará a XML mediante JAXB.
- **Aplicación móvil**. En el desarrollo de esta tarea se ha implementado la aplicación móvil en BlackBerry y Android, estableciendo los componentes principales de la aplicación: El módulo de ejecución de comandos y el módulo de conexión a los servicios. El módulo de ejecución de comandos es el encargado de actuar sobre el teléfono para ejecutar el comando recibido. El módulo de conexión con el servicio se encarga de obtener el comando estructurado de respuesta a partir del análisis de reconocimiento de voz. Ambos módulos tendrán una definición idéntica en las plataformas seleccionadas, variando la implementación en lo que se refiere a la interacción con la plataforma móvil. El resto de la aplicación cliente

está compuesta básicamente por la interfaz de usuario, absolutamente dependiente y diferente, incluso a nivel conceptual, en cada una de las plataformas. Esta tarea se centró en el desarrollo de la interfaz de usuario y el módulo de ejecución de comandos, dejando el módulo encargado de la conexión con el servicio.

- **Sistema de comunicación entre módulos**. El módulo de conexión del servicio tiene como misión de gestionar las conexiones a los servicios y servidores web que se encargarán de, por un lado transformar el comando hablado en texto plano y por otro convertir este texto plano en un comando estructurado basado en tecnología XML que será proporcionado al módulo de ejecución de comandos. El funcionamiento del sistema comienza con el dictado del comando, como por ejemplo: "Inserta una reunión a las 9 el próximo martes en el CEEIM con Miguel Ángel". Después la aplicación terminal compone un mensaje con este audio para enviárselo al Servidor ASR que recibirá el mensaje y lo traducirá en texto enviado de vuelta a la aplicación móvil. Posteriormente, este texto será enviado al servidor que implementa el motor de NLP que recibirá el mensaje y lo transformará en un documento XML que describe el comando a ejecutar.

V.2. LÍNEAS FUTURAS

Como se ha comentado en el apartado anterior, los resultados de este trabajo han quedado superados por otros sistemas de amplia difusión en la actualidad que no existían cuando se comenzó el desarrollo de este trabajo.

Sin embargo, eso no quiere decir que este trabajo quede abandonado y no pueda continuarse, sino al contrario. Las tecnologías empleadas han sufrido amplias mejoras en muy poco tiempo, como es el caso del reconocimiento de voz, que ha mejorado significativamente y ahora es ampliamente utilizado y bien aceptado por los usuarios de *smartphones*.

Además, todavía queda mucho trabajo por hacer desde el punto de vista de tener asistentes NLP, como por ejemplo, que el asistente sea capaz de guardar un estado de la conversación con el usuario para que el sistema pueda "dialogar" con el usuario para realizar acciones más complejas y específicas.

Con todos estos puntos, la opción más clara como desarrollo futuro es adaptar la tecnología a aplicaciones específicas que otras soluciones más genéricas no puedan cumplir. Incluso que esas aplicaciones se ejecuten en entornos que no tengan porque ser los dispositivos móviles. En concreto, los resultados de este trabajo se han empleado para el desarrollo de un asistente telefónico inteligente capaz de cerrar citas que se ha denominado Voozer. Este sistema emplea la misma tecnología que la descrita en este documento con la salvedad del reconocimiento de voz, donde se ha utilizado la solución INVOX desarrollada por Vócali. En este caso, igual que en el trabajo planteado en esta tesis, todo el sistema estaba en la nube, pero en este caso no existía aplicación móvil ya que el asistente se ha desarrollado para telefonía convencional, por lo que se trata de un sistema IVR.

La idea del sistema Voozer es muy sencilla. Voozer está conectado con la base de datos de clientes de una determinada empresa y puede funcionar de una forma reactiva o proactiva. En el primer caso, el usuario llama a un teléfono para pedir una cita (p.e. una clínica dental) y el sistema se encarga de cerrar la cita automáticamente teniendo en cuenta las agendas y horarios correspondientes (siguiendo el ejemplo, consultando en las agendas de los dentistas y en el horario de la clínica). En el caso proactivo, el sistema puede programarse para ver qué pacientes no han venido en más de un año para llamarlos a todos e intentar concertar una cita de forma automática. Este asistente trabaja con expresiones temporales al igual que el trabajo expuesto en esta tesis, por lo que tiene que ser capaz de resolver expresiones temporales de diverso tipo (difusas, inexactas,…) en lenguaje natural para convertirlas a un valor concreto. Además, también existe el caso de ambigüedades asociadas al lenguaje que deben resolverse en tiempo real para que el asistente pueda continuar la conversación.

Esto es sólo un ejemplo de un asistente que emplea la tecnología desarrollada en este trabajo. Por otro lado, con el éxito de Apple Siri, Google Now o Microsoft Cortana, han aparecido (y aparecen continuamente) multitud de empresas y nuevos productos relacionados con los asistentes virtuales. Muchas de ellas no ofrecen un producto en cuestión sino un *framework* o APIs para poder crear uno mismo su propio asistente sin tener conocimientos en técnicas de reconocimiento de voz (ASR), conversión texto a voz (TTS) o procesamiento de lenguaje natural (NLP). Esto implica que hay hacer un trabajo de vigilancia tecnológica para ver qué ofrecen estas nuevas soluciones y ver qué

aspectos se pueden mejorar, porque en un momento determinado puede ser más interesante utilizar ciertos módulos de estos nuevos productos y aplicar sólo los módulos desarrollados en este trabajo del que carezcan estos otros sistemas.

Uno de los asistentes que más éxito ha tenido fuera de los creados por multinacionales como Apple o Google, es Sherpa (http://sher.pa), que es de creación español y que admite bastante de las funcionalidades que se han descrito en este trabajo, aunque no dispone de un sistema de análisis y resolución de expresiones temporales tan potente como el descrito en este trabajo, y los mecanismos de desambiguación so bastante pobres, seguramente debido a que el sistema está pensado para un ámbito más general, lo que complica la desambiguación. En cualquier caso, Sherpa es un potente producto que ha conseguido que una empresa como Samsung preinstale este software en sus *smartphones*.

Otro sistema a destacar es wit.ai, que ha sido adquirido recientemente por Facebook y que se describe como una API de lenguaje natural que permite a los desarrolladores crear aplicaciones que empleén lenguaje natural para domótica, *wearables*, robots, asistentes, etc. Esta API incluye también tecnología de reconocimiento de voz y, aunque no dicen si es propia o de terceros, por investigaciones en diferentes portales especializados en reconocimiento de voz se sabe que están usando los motores de Sphinx y Kaldi, dos de las soluciones *open source* más importantes de ASR en la actualidad.

Lógicamente, no se puede competir con sistemas como Apple Siri o Google Now porque son productos con presupuestos millonarios (Siri había alcanzado alrededor de 23,5 millones de dólares para desarrollar el sistema antes de que fuese comprada por Apple) pero si que se pueden desarrollar sistemas específicos para nichos de mercado.

ANEXO I. EJEMPLOS DE COMANDOS XML

Ejemplos de comandos del calendario

A continuación se muestran distintos ejemplos de órdenes en lenguaje natural y el comando XML que se debería generar por este motor de procesamiento del lenguaje natural.

Ejemplo 1: "Inserta en la agenda una cita con pedro vivancos el próximo lunes por la mañana"

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:AddCalendarEvent
xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0
http://vocali.dyndns.org/smsmagic/1.0/smsmagic.xsd"
xmlns:ns2="http://www.vocali.net/smsmagic/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <startDay>2009-11-09</startDay>
    <startTime>09:00:00</startTime>
    <endDay xsi:nil="true"/>
    <endTime xsi:nil="true"/>
    <place xsi:nil="true"/>
    <repeatEvery xsi:nil="true"/>
    <repetitionsNumber xsi:nil="true"/>
    <description>Inserta en la agenda una cita con pedro vivancos el
próximo lunes por la mañana</description>
    <guests>
        <guest name="pedro vivancos"/>
    </guests>
</ns2:AddCalendarEvent>
```

Ejemplo 2: "Inserta una reunión a las 5 de la tarde del próximo martes con Rafael Valencia en el INFO"

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:AddCalendarEvent
xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0
http://vocali.dyndns.org/smsmagic/1.0/smsmagic.xsd"
xmlns:ns2="http://www.vocali.net/smsmagic/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <startDay>2009-11-10</startDay>
```

```
    <startTime>17:00:00</startTime>
    <endDay xsi:nil="true"/>
    <endTime xsi:nil="true"/>
    <place>INFO</place>
    <repeatEvery xsi:nil="true"/>
    <repetitionsNumber xsi:nil="true"/>
    <description>Inserta una reunión a las 5 de la tarde del próximo
martes con Rafael Valencia en el INFO</description>
    <guests>
        <guest name="Rafael Valencia"/>
    </guests>
</ns2:AddCalendarEvent>
```

Ejemplo 3: "Inserta en el calendario que mañana tenemos una reunión con INDRA a las 10 de la mañana"

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:AddCalendarEvent
xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0
http://vocali.dyndns.org/smsmagic/1.0/smsmagic.xsd"
xmlns:ns2="http://www.vocali.net/smsmagic/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <startDay>2009-11-07</startDay>
    <startTime>10:00:00</startTime>
    <endDay xsi:nil="true"/>
    <endTime xsi:nil="true"/>
    <place xsi:nil="true"/>
    <repeatEvery xsi:nil="true"/>
    <repetitionsNumber xsi:nil="true"/>
    <description>Inserta en el calendario que mañana tenemos una
reunión con INDRA a las 10 de la mañana</description>
    <guests>
      <guest name="INDRA"/>
    </guests>
</ns2:AddCalendarEvent>
```

Ejemplos de comandos de la agenda

A continuación se muestran distintos ejemplos de órdenes en lenguaje natural y el comando XML que se debería generar por este motor de procesamiento del lenguaje natural.

Ejemplo 1: "Inserta a Juan Márquez de la empresa SOMOS con teléfono 968584858 y email juan.márquez@somos.es"

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:AddContact xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0
http://vocali.dyndns.org/smsmagic/1.0/smsmagic.xsd"
xmlns:ns2="http://www.vocali.net/smsmagic/1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
        <name>Juan Márquez</name>
    <phone>968584858</phone>
    <email>juan.márquez@somos.es</email>
    <company>SOMOS</company>
</ns2:AddContact>
```

Ejemplo 2: "Inserta el contacto Javier Martínez 999999999 javier.martinez@indra.es"

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:AddContact xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0
http://vocali.dyndns.org/smsmagic/1.0/smsmagic.xsd"
xmlns:ns2="http://www.vocali.net/smsmagic/1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
        <name>Javier Martínez</name>
    <phone>999999999</phone>
    <email>javier.martinez@indra.es</email>
    <company xsi:nil="true"/>
</ns2:AddContact>
```

Ejemplo 3: "Añade en la agenda a Manuel Cánovas con teléfono 696 758 214"

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:AddContact xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0
http://vocali.dyndns.org/smsmagic/1.0/smsmagic.xsd"
xmlns:ns2="http://www.vocali.net/smsmagic/1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
        <name>Manuel Cánovas</name>
    <phone>696758214</phone>
    <email xsi:nil="true"/>
    <company xsi:nil="true"/>
</ns2:AddContact>
```

Ejemplos de comandos de email

A continuación se muestran distintos ejemplos de órdenes en lenguaje natural y el comando XML que se debería generar por este motor de procesamiento del lenguaje natural.

Ejemplo 1: "Crear un nuevo correo electrónico para Javier Pita con asunto pruebas y cuerpo Hola Javier, el informe de pruebas te lo enviaremos en los próximos días Gracias un saludo."

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:SendEmail xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0
http://vocali.dyndns.org/smsmagic/1.0/smsmagic.xsd"
xmlns:ns2="http://www.vocali.net/smsmagic/1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
        <sendTo>Javier Pita</sendTo>
        <subject>pruebas</subject>
        <message>Hola Javier, el informe de pruebas te lo enviaremos en los próximos días Gracias un
saludo.</message>
</ns2:SendEmail
```

Ejemplo 2: "Manda un email a Alfredo Villalba texto buenos días Alfredo ¿cuándo nos podríamos ver?"

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:SendEmail xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0
http://vocali.dyndns.org/smsmagic/1.0/smsmagic.xsd"
xmlns:ns2="http://www.vocali.net/smsmagic/1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
        <sendTo>Alfredo Villalba</sendTo>
        <subject>buenos días Alfre...</subject>
        <message>buenos días Alfredo ¿cuándo nos podríamos ver?</message>
</ns2:SendEmail>
```

Ejemplo 3: "Crea el nuevo correo para Paco Flórez con asunto Sobre contacto de Loquendo y texto Hola Paco, este es el contacto de Loquendo en España. Marco Piña Sánchez. +34 618785956. marco.pinasanchez@loquendo.com. Saludos"

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:SendEmail xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0
http://vocali.dyndns.org/smsmagic/1.0/smsmagic.xsd"
xmlns:ns2="http://www.vocali.net/smsmagic/1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
        <sendTo>Paco Flórez</sendTo>
        <subject>Sobre contacto de Loquendo</subject>
        <message>Hola Paco, este es el contacto de Loquendo en España. Marco Piña Sánchez. +34
618785956. marco.pinasanchez@loquendo.com. Saludos</message>
</ns2:SendEmail>
```

Ejemplos de comandos de SMS

A continuación se muestran distintos ejemplos de órdenes en lenguaje natural y el comando XML que se debería generar por este motor de procesamiento del lenguaje natural.

Ejemplo 1: "Envía un SMS al teléfono 639011266 que diga: por favor confírmame que has recibido el correo electrónico que te envié."

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:SendSms xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0
http://vocali.dyndns.org/smsmagic/1.0/smsmagic.xsd"
xmlns:ns2="http://www.vocali.net/smsmagic/1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
        <sendTo>639011266</sendTo>
        <message>por favor confírmame que has recibido el correo electrónico que te
envié.</message>
</ns2:SendSms>
```

Ejemplo 2: "Envia un SMS ugentemente que diga llamame cuando puedas. Es para ver el tema pendiente."

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:SendSms xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0
http://vocali.dyndns.org/smsmagic/1.0/smsmagic.xsd"
```

```
xmlns:ns2="http://www.vocali.net/smsmagic/1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <sendTo xsi:nil="true"/>
        <message>ugentemente que diga llamame cuando puedas. Es para ver el tema pendiente.</message>
    </ns2:SendSms>
```

Ejemplo 3: "Crear SMS con el cuerpo al final de la mañana pasame un los precios"

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:SendSms xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0 http://vocali.dyndns.org/smsmagic/1.0/smsmagic.xsd" xmlns:ns2="http://www.vocali.net/smsmagic/1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <sendTo xsi:nil="true"/>
        <message>al final de la mañana pasame un los precios</message>
</ns2:SendSms>
```

Ejemplos de comandos de tareas

A continuación se muestran distintos ejemplos de órdenes en lenguaje natural y el comando XML que se debería generar por este motor de procesamiento del lenguaje natural.

Ejemplo 1: "Tarea en los proximos dias tengo que enviar presupuesto a Neosistec"

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:AddTask xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0 http://vocali.dyndns.org/smsmagic/1.0/smsmagic.xsd" xmlns:ns2="http://www.vocali.net/smsmagic/1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <description>en los proximos dias tengo que enviar presupuesto a Neosistec</description>
        <date>2010-03-16</date>
</ns2:AddTask>
```

Ejemplo 2: "Tarea completar información nuevos clientes de INVOX"

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:AddTask xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0 http://vocali.dyndns.org/smsmagic/1.0/smsmagic.xsd"
```

```
xmlns:ns2="http://www.vocali.net/smsmagic/1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <description>completar información nuevos clientes de INVOX</description>
        <date xsi:nil="true"/>
    </ns2:AddTask>
```

Ejemplo 3: "Añade una tarea para recordarme que tengo que enviar la información a los portugueses."

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:AddTask xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0 http://vocali.dyndns.org/smsmagic/1.0/smsmagic.xsd" xmlns:ns2="http://www.vocali.net/smsmagic/1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <description>para recordarme que tengo que enviar la información a los portugueses.</description>
        <date xsi:nil="true"/>
</ns2:AddTask>
```

Ejemplos de comandos de notas

A continuación se muestran distintos ejemplos de órdenes en lenguaje natural y el comando XML que se debería generar por este motor de procesamiento del lenguaje natural.

Ejemplo 1: "Nota titulo reunion Soni texto recordar incluir en la presentacion es el estado de las lineas activadas."

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:AddMemo xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0 http://vocali.dyndns.org/smsmagic/1.0/smsmagic.xsd" xmlns:ns2="http://www.vocali.net/smsmagic/1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <title>reunion Soni</title>
    <memo>recordar incluir en la presentacion es el estado de las lineas activadas.</memo>
</ns2:AddMemo>
```

Ejemplo 2: "Nota con título Tareas necesarias a realizar esta tarde que dice Es importante que revise las líneas internas de oficina vodafone."

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

<ns2:AddMemo xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0 http://vocali.dyndns.org/smsmagic/1.0/smsmagic.xsd" xmlns:ns2="http://www.vocali.net/smsmagic/1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<title>Tareas necesarias a realizar esta tarde</title>
<memo>Es importante que revise las líneas internas de oficina vodafone.</memo>
</ns2:AddMemo>

Ejemplo 3: "Insertar nota he leído un artículo en la sección de economía de El País que habla que el mercado de aplicaciones móviles va a crecer un 800% hasta 2013"

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:AddMemo xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0 http://vocali.dyndns.org/smsmagic/1.0/smsmagic.xsd" xmlns:ns2="http://www.vocali.net/smsmagic/1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<title>he leído un artíc...</title>
<memo>he leído un artículo en la sección de economía de El País que habla que el mercado de aplicaciones móviles va a crecer un 800% hasta 2013</memo>
</ns2:AddMemo>

ANEXO II. COMANDOS DE VOZ IMPLEMENTADOS

En este anexo se muestran los diferentes comandos de voz implementados en este trabajo para controlar diferentes aplicaciones existentes en los *smartphones.* En cada comando se muestra una pequeña descripción del mismo y ejemplos donde se puede ver ante un determinado comando cuál es la salida en formato XML que ofrece el sistema. Este XML se basa en un XML Schema que se añade como un apartado de este capítulo.

CALENDARIO DE EVENTOS

COMANDO ADDCALENDAREVENT

Añade una nueva cita en el calendario. Este comando está formado por los distintos elementos:

Elemento	Obligatorio	Tipo	Descripción
startDay	Sí	xsd:date	Fecha (día) de inicio del evento.
startTime	No	xsd:time	Hora de inicio del evento.
endDay	No	xsd:date	Fecha (día) de fin del evento.
endTime	No	xsd:time	Hora de fin del evento.
place	No	xsd:string	Lugar del evento.
repeatEvery	No	xsd:int	Indica cada cuantos días se debe repetir el evento.
repetitionNumber	No	xsd:int	Indica cuantas veces se debe repetir el evento.
guests	No	(Ver apéndice 1)	Personas que asistirán al evento.
description	No	xsd:string	Descripción del evento. Al menos para el prototipo será el propio cuerpo del mensaje.

Ejemplos:

1. "Inserta una reunión a las 9 el próximo martes en el CEEIM con Jesualdo".

```
<?xml version="1.0" encoding="UTF-8"?>
<vocali:AddCalendarEvent
xmlns:vocali="http://www.vocali.net/smsmagic/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0
smsmagic.xsd">
    <startDay>2009-10-13</startDay>
    <startTime>09:00:00</startTime>
    <endDay xsi:nil="true"/>
    <endTime xsi:nil="true"/>
    <place>CEEIM</place>
    <repeatEvery xsi:nil="true"/>
    <repetitionsNumber xsi:nil="true"/>
    <description>Inserta una reunión a las 9 el próximo martes en
el CEEIM con Jesualdo</description>
    <guests>
        <guest name="Jesualdo"/>
    </guests>
</vocali:AddCalendarEvent>
```

2. "Registra que tengo curso de inglés todos los martes de noviembre a las 8 de la tarde".

```
<?xml version="1.0" encoding="UTF-8"?>
<vocali:AddCalendarEvent
xmlns:vocali="http://www.vocali.net/smsmagic/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0
smsmagic.xsd">
    <startDay>2009-10-13</startDay>
    <startTime>20:00:00</startTime>
    <endDay xsi:nil="true"></endDay>
    <endTime xsi:nil="true"></endTime>
    <place xsi:nil="true"></place>
    <repeatEvery>7</repeatEvery>
    <repetitionsNumber>4</repetitionsNumber>
```

```
    <description>Registra que tengo curso de inglés todos los
martes de noviembre a las 8 de la tarde</description>
    <guests/>
```

COMANDO REMOVECALENDAREVENT

Elimina una cita del calendario. Este comando está formado por los distintos elementos:

Elemento	Obligatorio	Tipo	Descripción
startDay	Sí	xsd:date	Fecha (día) de inicio del evento.
startTime	No	xsd:time	Hora de inicio del evento.
place	No	xsd:string	Lugar del evento.

Ejemplos:

- "Elimina la reunión del próximo martes en el CEEIM".

```
<?xml version="1.0" encoding="UTF-8"?>
<vocali:RemoveCalendarEvent
xmlns:vocali="http://www.vocali.net/smsmagic/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0
smsmagic.xsd">
    <startDay>2009-10-13</startDay>
    <startTime xsi:nil="true"></startTime>
    <place>CEEIM</place>
</vocali:RemoveCalendarEvent>
```

- "Cancela la reunión del martes 3 de noviembre a las 8 de la tarde".

```
<?xml version="1.0" encoding="UTF-8"?>
<vocali:RemoveCalendarEvent
xmlns:vocali="http://www.vocali.net/smsmagic/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0
smsmagic.xsd">
    <startDay>2009-11-03</startDay>
    <startTime>20:00:00</startTime>
```

```
    <place xsi:nil="true"></place>
</vocali:RemoveCalendarEvent>
```

COMANDO MODIFYCALENDAREVENT

Modifica un evento del calendario. Este comando está formado por los distintos elementos:

Elemento	Obligatorio	Tipo	Descripción
startDay	Sí	xsd:date	Fecha (día) de inicio del evento que va a ser modificado.
startTime	No	xsd:time	Hora de inicio del evento que va a ser modificado.
place	No	xsd:string	Lugar del evento que va a ser modificado.
newStartDay	Sí	xsd:date	Nueva fecha (día) de inicio del evento.
newStartTime	No	xsd:time	Nueva hora de inicio del evento.
newEndTime	No	xsd:time	Nueva hora de fin del evento.
newPlace	No	xsd:string	Nuevo lugar de celebración del evento.
newDescription	No	xsd:string	Descripción del evento. Para el prototipo será el propio texto de mensaje que sería interesante adjuntar con la descripción original.

Ejemplos:

- "Cambia la reunión del próximo martes a las 9 a las 11".

```
<?xml version="1.0" encoding="UTF-8"?>
<vocali:ModifyCalendarEvent
xmlns:vocali="http://www.vocali.net/smsmagic/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0
smsmagic.xsd">
    <startDay>2009-10-13</startDay>
    <startTime>09:00:00</startTime>
```

```
    <place  xsi:nil="true"/>
    <newStartDay>2009-10-13</newStartDay>
    <newStartTime>11:00:00</newStartTime>
    <newEndTime xsi:nil="true"/>
    <newPlace xsi:nil="true"/>
    <newDescription xsi:nil="true"/>
</vocali:ModifyCalendarEvent>
```

AGENDA DE CONTACTOS

COMANDO ADDCONTACT

Añade a una persona en la agenda de contactos. Este comando está formado por los distintos elementos:

Elemento	Obligatorio	Tipo	Descripción
name	Sí	xsd:string	Nombre completo del contacto.
phone	No	xsd:string	Número de teléfono del contacto.
email	No	xsd:string	Correo electrónico del contacto.
company	No	xsd:string	Empresa del contacto.

Ejemplos:

1. "Añade en la agenda a Antonio García Sánchez con el teléfono 555 111 444".

```
<?xml version="1.0" encoding="UTF-8"?>
<vocali:AddContact
xmlns:vocali="http://www.vocali.net/smsmagic/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0
smsmagic.xsd">
    <name>Antonio García Sánchez</name>
    <phone>555111444</phone>
    <email xsi:nil="true"/>
    <company xsi:nil="true"/>
</vocali:AddContact>
```

COMANDO REMOVECONTACT

Elimina una entrada de la agenda de contactos. Este comando está formado por los distintos elementos:

Elemento	Obligatorio	Tipo	Descripción
name	Sí	xsd:string	Nombre completo del contacto.

Ejemplos:

1. "Elimina de la agenda a Antonio García Sánchez".

```
<?xml version="1.0" encoding="UTF-8"?>
<vocali:RemoveContact
xmlns:vocali="http://www.vocali.net/smsmagic/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0
smsmagic.xsd">
    <name>Antonio García Sánchez</name>
</vocali:RemoveContact>
```

COMANDO MODIFYCONTACT

Descripción: elimina una entrada de la agenda de contactos. Este comando está formado por los distintos elementos:

Elemento	Obligatorio	Tipo	Descripción
name	Sí	xsd:string	Nombre completo del contacto.
newName	No	xsd:string	Nuevo nombre del contacto.
newPhone	No	xsd:string	Nuevo teléfono del contacto.
newEmail	No	xsd:string	Nuevo correo electrónico del contacto.
newCompany	No	xsd:string	Nueva empresa del contacto.

Ejemplos:

1. "Cambia el email de Antonio García Sánchez por agarcia@acme.com".

```
<?xml version="1.0" encoding="UTF-8"?>
<vocali:ModifyContact
```

```
xmlns:vocali="http://www.vocali.net/smsmagic/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0
smsmagic.xsd">
    <name>Antonio García Sánchez</name>
    <newName xsi:nil="true"/>
    <newPhone xsi:nil="true"/>
    <newEmail>agarcia@acme.com</newEmail>
    <newCompany xsi:nil="true"/>
</vocali:ModifyContact>
```

SISTEMA DE ALARMA/DESPERTADOR

COMANDO SETALARMCLOCK

Establece la alarma. Este comando está formado por los distintos elementos:

Elemento	Obligatorio	Tipo	Descripción
date	N	xsd:date	Fecha (día) para establecer la alarma.
time	Sí	xsd:time	Hora a la que establecer la alarma.

Ejemplos:

1. "Pon la alarma mañana a las 9 de la mañana"

```
<?xml version="1.0" encoding="UTF-8"?>
<vocali:SetAlarmClock
xmlns:vocali="http://www.vocali.net/smsmagic/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0
smsmagic.xsd">
    <date>2009-10-09</date>
    <time>09:00:00</time>
</vocali:SetAlarmClock>
```

2. "Pon la alarma a las 9"

```
<?xml version="1.0" encoding="UTF-8"?>
<vocali:SetAlarmClock
xmlns:vocali="http://www.vocali.net/smsmagic/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0
```

```
smsmagic.xsd">
    <date xsi:nil="true"/>
    <time>09:00:00</time>
</vocali:SetAlarmClock>
```

COMANDO UNSETALARMCLOCK

Desactiva la alarma. Este comando está formado por los distintos elementos:

Elemento	Obligatorio	Tipo	Descripción
date	No	xsd:date	Fecha (día) a la que está establecida la alarma
time	No	xsd:dateTime	Hora a la que está establecida la alarma.

Ejemplos:

1. "Quita la alarma de mañana a las 9".

```
<?xml version="1.0" encoding="UTF-8"?>
<vocali:UnsetAlarmClock
xmlns:vocali="http://www.vocali.net/smsmagic/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0
smsmagic.xsd">
    <date>2009-10-09</date>
    <time>09:00:00</time>
</vocali:UnsetAlarmClock>
```

2. "Desactiva la alarma".

```
<?xml version="1.0" encoding="UTF-8"?>
<vocali:UnsetAlarmClock
xmlns:vocali="http://www.vocali.net/smsmagic/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0
smsmagic.xsd">
    <date xsi:nil="true"></date>
    <time xsi:nil="true"></time>
</vocali:UnsetAlarmClock>
```

COMANDO MODIFYALARMCLOCK

Modifica una alarma ya existente. Este comando está formado por los distintos elementos:

Elemento	Obligatorio	Tipo	Descripción
date	No	xsd:date	Fecha (día) a la que está establecida la alarma
time	No	xsd:dateTime	Hora a la que está establecida la alarma.
newDate	No	xsd:date	Nueva fecha (día) a la que establer la alarma
newTime	No	xsd:dateTime	Hora a la que establer la alarma.

Ejemplos:

1. "Cambia la alarma de mañana a las 10".

```
<?xml version="1.0" encoding="UTF-8"?>
<vocali:ModifyAlarmClock
xmlns:vocali="http://www.vocali.net/smsmagic/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0
smsmagic.xsd">
    <date>2009-10-09</date>
    <time xsi:nil="true"/>
    <newDate>2009-10-09</newDate>
    <newTime>10:00:00</newTime>
</vocali:UnsetAlarmClock>
```

2. "Cambia la alarma de las 9 a las 10"

```
<?xml version="1.0" encoding="UTF-8"?>
<vocali:ModifyAlarmClock
xmlns:vocali="http://www.vocali.net/smsmagic/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.vocali.net/smsmagic/1.0
smsmagic.xsd">
    <date xsi:nil="true"></date>
    <time>09:00:00</time>
    <newDate></newDate>
    <newTime>10:00:00</newTime>
</vocali:UnsetAlarmClock>
```

GESTIÓN DE COMANDOS NO LEGIBLES

Es posible que lleguen textos al servicio web que no tendrán nada que ver con ninguna de las operaciones definidas en el ámbito del proyecto.

COMANDO NOACTIONRECOGNIZED

Indica que no se ha podido obtener ningún comando del texto recibido. Está formado por los siguientes elementos:

Elemento	Obligatorio	Tipo	Descripción
code	Sí	xsd:int	Código numérico que identifica el error ocurrido.
description	No	xsd:string	Descripción de la incidencia.

Nota: códigos de error definitivos por definir.

XML SCHEMA EMPLEADO

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema xmlns:vocali="http://www.vocali.net/smsmagic/1.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.vocali.net/smsmagic/1.0">
    <xsd:element name="AddCalendarEvent">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="startDay" type="xsd:date"
nillable="false"/>
                <xsd:element name="startTime" type="xsd:time"
nillable="true"/>
                <xsd:element name="endDay" type="xsd:date" nillable="true"/>
                <xsd:element name="endTime" type="xsd:time" nillable="true"/>
                <xsd:element name="place" type="xsd:string" nillable="true"/>
                <xsd:element name="repeatEvery" type="xsd:int"
nillable="true"/>
                <xsd:element name="repetitionsNumber" type="xsd:int"
nillable="true"/>
                <xsd:element name="description" type="xsd:string"
nillable="true"/>
                <xsd:element name="guests" nillable="true">
                    <xsd:complexType>
                        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
                            <xsd:element name="guest">
                                <xsd:complexType>
                                    <xsd:attribute name="name"
```

```
type="xsd:string"/>
                                    </xsd:complexType>
                                </xsd:element>
                            </xsd:sequence>
                        </xsd:complexType>
                    </xsd:element>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="RemoveCalendarEvent">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="startDay" type="xsd:date"
nillable="false"/>
                    <xsd:element name="startTime" type="xsd:time"
nillable="true"/>
                    <xsd:element name="place" type="xsd:string" nillable="true"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="ModifyCalendarEvent">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="startDay" type="xsd:date"
nillable="false"/>
                    <xsd:element name="startTime" type="xsd:time"
nillable="true"/>
                    <xsd:element name="place" type="xsd:string" nillable="true"/>
                    <xsd:element name="newStartDay" type="xsd:date"
nillable="false"/>
                    <xsd:element name="newStartTime" type="xsd:time"
nillable="true"/>
                    <xsd:element name="newEndTime" type="xsd:time"
nillable="true"/>
                    <xsd:element name="newPlace" type="xsd:string"
nillable="true"/>
                    <xsd:element name="newDescription" type="xsd:string"
nillable="true"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="AddContact">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="name" type="xsd:string" nillable="false"/>
                    <xsd:element name="phone" type="xsd:string" nillable="true"/>
```

```
                <xsd:element name="email" type="xsd:string" nillable="true"/>
                <xsd:element name="company" type="xsd:string"
nillable="true"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="RemoveContact">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="name" type="xsd:string" nillable="false"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="ModifyContact">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="name" type="xsd:string" nillable="false"/>
                <xsd:element name="newName" type="xsd:string"
nillable="true"/>
                <xsd:element name="newPhone" type="xsd:string"
nillable="true"/>
                <xsd:element name="newEmail" type="xsd:string"
nillable="true"/>
                <xsd:element name="newCompany" type="xsd:string"
nillable="true"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="SetAlarmClock">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="date" type="xsd:date" nillable="true"/>
                <xsd:element name="time" type="xsd:time" nillable="false"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="UnsetAlarmClock">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="date" type="xsd:date" nillable="true"/>
                <xsd:element name="time" type="xsd:time" nillable="true"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="ModifyAlarmClock">
        <xsd:complexType>
```

```
            <xsd:sequence>
                <xsd:element name="date" type="xsd:date" nillable="false"/>
                <xsd:element name="time" type="xsd:time" nillable="true"/>
                <xsd:element name="newDate" type="xsd:date" nillable="true"/>
                <xsd:element name="newTime" type="xsd:time" nillable="true"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="NoActionRecognized">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="code" type="xsd:int" nillable="false"/>
                <xsd:element name="description" type="xsd:string"
nillable="true"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

Figura 75. Fichero XML Schema (XSD) empleado.

ANEXO III. TEST SOCIO-LINGÜÍSTICO

Imagina que pudieras controlar mediante la voz algunas de las aplicaciones más típicas incluidas en un Smartphone o teléfono móvil, como por ejemplo el teléfono, SMS, agenda, calendario y correo electrónico.

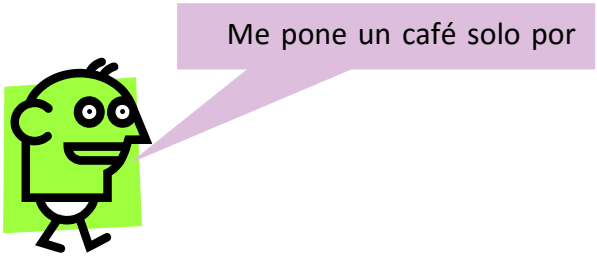
Con sólo unas palabras sería posible que estas aplicaciones hicieran lo que tú les estas pidiendo, como si realmente te entendieran.

El objetivo de esta encuesta es determinar cuáles son las expresiones que te gustaría utilizar para "comunicarte" con tu dispositivo móvil, de manera que te resulte lo más cómodo posible.

Para ello te proponemos una serie de acciones que se han automatizado y que es posible controlar mediante un comando de voz.

En cada imagen se presenta una situación concreta, con un breve texto que la describe, junto con un personaje, al que llamaremos Pepe, que es el que debe ordenar o pedir que se realice la acción. Si tú fueras esa persona ¿qué te gustaría decir en cada situación?

Por ejemplo, Pepe se acaba de levantar y se dirige al bar de la esquina para tomar un café, una vez en el bar el camarero llega y Pepe hace su petición. Existen varias opciones a la hora de pedir un café.



Pero Pepe también podría haber dicho:

Un café por favor

Un solo

Me gustaría tomar un café

Quiero un café

…

A continuación se muestran las aplicaciones del dispositivo móvil que Pepe puede manejar mediante la voz.

Ahora se plantearán una serie de situaciones relacionadas con estas aplicaciones en donde Pepe les puede hablar para conseguir lo que desea.

1º PARTE

Las expresiones que se podrían utilizar para que estos elementos funcionen son muchas, así que puedes escribir todas las que consideres.

1. Pepe quiere interactuar con la aplicación 

. ¿Qué podría decirle al dispositivo móvil para hacer que dicha aplicación funcione y realice la funcionalidad deseada?

Otras sugerencias:

2. Pepe quiere interactuar con la aplicación  . ¿Qué podría decirle al dispositivo móvil para hacer que dicha aplicación funcione y realice la funcionalidad deseada?

Otras sugerencias:

3. Pepe quiere interactuar con la aplicación  . ¿Qué podría decirle al dispositivo móvil para hacer que dicha aplicación funcione y realice la funcionalidad deseada?

Otras sugerencias:

4. Pepe quiere interactuar con la aplicación 

. ¿Qué podría decirle al dispositivo móvil para hacer que dicha aplicación funcione y realice la funcionalidad deseada?

Otras sugerencias:

5. Pepe quiere interactuar con la aplicación 
. ¿Qué podría decirle al dispositivo móvil para hacer que dicha aplicación funcione y realice la funcionalidad deseada?

Otras sugerencias:

6. Pepe quiere interactuar con la aplicación  . ¿Qué podría decirle al dispositivo móvil para hacer que dicha aplicación funcione y realice la funcionalidad deseada?

Otras sugerencias:

7. Pepe quiere interactuar con la aplicación  . ¿Qué podría decirle al dispositivo móvil para hacer que dicha aplicación funcione y realice la funcionalidad deseada?

Otras sugerencias:

8. Pepe quiere interactuar con la aplicación  . ¿Qué podría decirle al dispositivo móvil para hacer que dicha aplicación funcione y realice la funcionalidad deseada?

Otras sugerencias:

2º PARTE

Aunque el dispositivo móvil de Pepe es bastante listo, puede ocurrir que algunas aplicaciones no funcionen correctamente cuando se les pide algo.

BLA BLA BLA

Esto puede pasar porque el dispositivo no ha escuchado lo que Pepe ha dicho o bien porque, aunque lo haya entendido, esa orden se puede aplicar a varias aplicaciones y el sistema no sabe a cuál de ellos se está refiriendo.

Entonces, si tú fueras Pepe ¿qué te gustaría que pasara si la aplicación que quieres que funcionara no lo hiciera?

3º PARTE

¿Qué otras aplicaciones del dispositivo móvil te gustaría poder controlar con la voz?

¿Qué funcionalidades de esas aplicaciones?

REFERENCIAS

[1]. ACM Transactions on Intelligent Systems and Technology (ACM TIST)

[2]. Agirre, E., Ansa, O., Hovy, E., Martinez, D. (2000). Enriching very large ontologies using the WWW. In Proceedings of the Workshop on Ontology Construction of the European Conference of AI (ECAI-00)

[3]. Aguado de Cea, G., Álvarez de Mon Rego, I., & Pareja-Lora, A. (2003). Primeras aproximaciones a la anotación lingüístico-ontológica de documentos de Web Semántica: OntoTag. Revista Iberoamericana de Inteligencia Artificial, 18, 37-49.

[4]. Alani H., Kim S., Millard D. E., Weal M. J., Hall W., Lewis P.H., Shadbolt N. R. (2003). IEEE Intelligent Systems. January/February 2003 14-21

[5]. Alfonseca E., Manandhar S. (2002), An unsupervised method for general named entity recognition and automated concept discovery. In Proceedings of the 1st International Conference on General WordNet, Mysore, India

[6]. AllegroGraph. http://agraph.franz.com/

[7]. Álvarez Silva, M., Ramírez Cruz, Y. & Anaya Sánchez, H. (2009). RECONOCEDOR DE NOMBRES DE ENTIDADES PARA EL ESPAÑOL BASADO EN EXPRESIONES REGULARES Y APRENDIZAJE AUTOMÁTICO. Innovación Tecnológica. 14 (4).

[8]. Ananiadou, S. (1994). A Methodology for Automatic Term Recognition. In: Proceedings of the 15th Conference on Computational Linguistics - Volume 2. COLING '94. [Online]. 1994, Stroudsburg, PA, USA: Association for Computational Linguistics, pp. 1034-1038. Available from: http://dx.doi.org/10.3115/991250.991317.

[9]. Antoniou, G. & vanHarmelen, F. (2004). A Semantic Web Primer. Cambridge, MA, USA: MIT Press.

[10]. Assadi, H. (1999) Construction of a regional ontology from text and ist use within a ocumentary system. In N. Guarino (ed.), Formal Ontology in Information Systems, Proceedings of FOIS-98, Trento, Italy, 1999, pages 236–249

[11]. Bahl, L.R. & Mercer, R.L. (1976). Part of speech assignment by a statistical decision algorithm. In: IEEE International Symposium on Information Theory. 1976, pp. 88-89.

[12]. Baker, J.K. (1979). Trainable grammars for speech recognition. The Journal of the Acoustical Society of America. 65 (S1). p.pp. S132-S132.

[13]. Baker, J.M., Deng, L., Glass, J., Khudanpur, S., Lee, C.H., Morgan, N., O'Shaughnessy, D. (2009). "Research Developments and Directions in Speech Recognition and Understanding, Part 1". IEEE SIGNAL PROCESSING MAGAZINE. Retrieved May, 2010.

[14]. Barrasa, J., Gómez-Pérez, A. (2004). R2o, an extensible and se- mantically based database to ontology mapping language. In In Proceedings of the 2nd Workshop on Semantic Web and Databases, Toronto, (Can) Aug., pages [1069,1070].

[15]. Beckett, D. Turtle: Terse rdf triple language. Recurso disponible on-line (Último acceso Feb.12): http://www.w3.org/TeamSubmission/turtle/.

[16]. Beigi, H. (2007). Fundamentals of Speaker Recognition.

[17]. Beizer, B. (1990). Software Testing Techniques. International Thompson Computer Press.

[18]. Berners-Lee, T., Hendler, J. & Lassila, O. (2001). The semantic web. Scientific american. 284 (5). p.pp. 28-37.

[19]. Berry, P.M., Melinda, G., Peinter, B., Yorke-Smith, N. (2011), "PTIME: Personalized assistance for calendaring", ACM TIST 2 (4), doi:10.1145/1989734.1989744

[20]. Bisson, G., Nedellec, C., Canamero, D. (2000) Designing Clustering Methods for Ontology Building - The Mo'K Workbench. In: S. Staab, A. Maedche, C. Nedellec, P. Wiemer-Hastings (eds.), Proceedings of the Workshop on Ontology Learning, 14th European Conference on Artificial Intelligence ECAI'00, Berlin, Germany

[21]. Bizer, C. (2003) D2r map - a database to rdf mapping language. In The 12th International World Wide Web Conference (www2003)

[22]. Bizer, C., Cyganiak, R., Garbers, J., Maresch, O., Becker, C. D2rq platform. Recurso on-line (Último acceso Feb.12): http://www4.wiwiss.fu-berlin.de/bizer/D2RQ/spec/.

[23]. Bizer, C. (2004) Triql: A query language for named graphs. Technical report, Freie Universitä¨t Berlin, Germany.

[24]. Bizer, C., Heath, T., Berners-Lee, T. (2009) Linked data - the story so far. Special Issue on Linked Data, International Journal on Semantic Web and Information Systems (IJSWIS), 5:[1,22], 2009.

[25]. Bliki Engine. Recurso disponible on-line (Último acceso Feb.12): http://code.google.com/p/gwtwiki/

[26]. Bojārs, U. & Breslin, J.G. (2007). ResumeRDF: Expressing skill information on the Semantic Web. In: 1 st International ExpertFinder Workshop. 2007.

[27]. Borst, W.N. (1997). Construction of Engineering Ontologies for Knowledge Sharing and Reuse. Centre for Telematics and Information Technology.

[28]. Boundy, D.E. (1991). "A taxonomy of programmers". ACM SIGSOFT Software Engineering Notes 16(4) 23-30. alcor.concordia.ca.

[29]. Bozsak, E., Ehrig, M., Handschuh, S., Hotho, A., Maedche, A., Motik, B., Oberle, D., Schmitz, C., Staab, S., Stojanovic, L., Stojanovic, N., Studer, R., Stumme, G., Sure, Y., Tane, J., Volz, R. & Zacharias, V. (2002). KAON - Towards a Large Scale Semantic Web. In: Proceedings of the Third International Conference on E-Commerce and Web Technologies. EC-WEB '02. [Online]. 2002, London, UK, UK: Springer-Verlag, pp. 304-313. Available from: http://dl.acm.org/citation.cfm?id=646162.680500. [Accessed: 28 July 2014].

[30]. Brank, J., Grobelnik, M. & Mladeni?, D. (2005). A survey of ontology evaluation techniques. In: In In Proceedings of the Conference on Data Mining and Data Warehouses (SiKDD 2005. 2005.

[31]. Brank, J., Grobelnik, M., & Mladeni?, D. (2005). A survey of ontology evaluation techniques. In In In Proceedings of the Conference on Data Mining and Data Warehouses (SiKDD 2005.

[32]. Brickley, D. & Miller, L. (2012). FOAF vocabulary specification 0.98. Namespace Document. 9.

[33]. Buneman, P. (1997). Semistructured Data. In: Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. PODS '97. [Online]. 1997, New York, NY, USA: ACM, pp. 117-121. Available from: http://doi.acm.org/10.1145/263661.263675. [Accessed: 31 May 2014].

[34]. CERN Website. http://home.web.cern.ch

[35]. CERN. The World Wide Web (WWW). http://line-mode.cern.ch/www/hypertext/WWW/TheProject.html

[36]. Chapman, S., Norton, B. & Ciravegna, F. (2005). Armadillo: Integrating knowledge for the semantic web. In: Proceedings of the Dagstuhl Seminar in Machine Learning for the Semantic Web. 2005.

[37]. Chomsky, N. (1956). Three models for the description of language. IRE Transactions on Information Theory. 2 (3). p.pp. 113-124.

[38]. Chowdhury, G.G. (2003). Natural language processing. Annual Review of Information Science and Technology. 37 (1). p.pp. 51-89.

[39]. Church, K.W. & Church, K.W. (1980). On Memory Limitations In Natural Language Processing.

[40]. Cimiano, P. & Völker, J. (2005). Towards large-scale, open-domain and ontology-based named entity classification. In: Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP). 2005.

[41]. Clark, Parsia. Pellet: Owl 2 reasoner for java. Recurso disponible on-line (Último acceso Feb.12): http://clarkparsia.com/pellet.

[42]. Corcho, O., Fernández-Lopez, M., Gómez-Pérez, A. (2003). Methodologies, tools and languages for building ontologies: Where is their meeting point? Data Knowl.Eng, 46(1):41–64.

[43]. Cuenca-Grau, B., Horrocks, I., Motik, B., Parsia, B., Patel−Schneider, P., Sattler, U. (2008) OWL 2: The Next Step for OWL. In Journal of Web Semantics. Vol. 6. No. 4. Pages 309−322.

[44]. Cullot, N., Ghawi, R., Yétongnon, K. Db2owl: A tool for automatic database-to-ontology mapping. Recurso dis- ponible on-line (Último acceso Feb.12): http://vision.u-bourgogne.fr/le2i/user_data/ publications/DB2OWL1.pdf.

[45]. Cunningham, H., Maynard, D., Bontcheva, K. & Tablan, V. (2002). GATE: an Architecture for Development of Robust HLT Applications. In: In Recent Advanced in Language Processing. 2002, pp. 168-175.

[46]. David Beckett & Tim Berners-Lee (2008). Turtle - Terse RDF Triple Language. [Online]. Available from: http://www.w3.org/TeamSubmission/2011/SUBM-turtle-20110328/. [Accessed: 30 May 2014].

[47]. Davis, R., Shrobe, H. & Szolovits, P. (1993). What Is a Knowledge Representation? AI Magazine. 14 (1). p.p. 17.

[48]. Deng, L., O'Shaughnessey, D. (2003). Speech Processing: A Dynamic and Optimization-Oriented Approach.

[49]. Devedzic, V. (2006). Semantic Web and Education (Integrated Series in Information Systems) (1st ed.). New York: Springer.

[50]. Devedzic, V., Gasevic, V. (2009). Web 2.0 & Semantic Web. Annals of Information Systems. Springer.

[51]. DIG Interface. http://dig.sourceforge.net/

[52]. Dou, D., Qin, H., LePendu, P. (2010) Ontograte: Towards automa- tic integration for relational databases and the semantic web through an ontology based framework. International Journal of Semantic Com- puting (IJSC), 4:[123,151], 2010.

[53]. Duan, Y., Cruz, C. (2011), Formalizing Semantic of Natural Language through Conceptualization from Existence. International Journal of Innovation, Management and Technology(2011) 2 (1), pp. 37-42.

[54]. Evers, R. (2005). Professional BlackBerry (1st ed.). Wrox. p. 308. ISBN 0764589539.

[55]. Faatz A., Steinmetz R. (2002). Ontology enrichment with texts from the WWW. Semantic Web Mining 2nd Workshop at ECML/PKDD-2002, 20th August 2002, Helsinki, Finland

[56]. Faure, D., Nedellec, C. (1998) A corpus-based conceptual clustering method for verb frames and ontology acquisition. In LREC workshop on Adapting lexical and corpus resources to sublanguages and applications, Granada, Spain

[57]. Fellbaum, F. (1998) WordNet: An Electronic Lexical Database. MIT Press, 1998.

[58]. Fernández-Breis, J.T. “Un Entorno para la Integración deOntologías para el Desarrollo de Sistemas para la Gestión deConocimiento”. Tesis Doctoral, Universidad de Murcia (2003)

[59]. Fernández, M., Gómez-Pérez, A., Juristo, N. (1997) METHONTOLOGY: from ontological art towards ontological engineering. In: Proceedings of AAAI97 spring symposium series, workshop on ontological engineering, Stanford, CA, pp 33–40.

[60]. Ferro, L., Gerber, L., Mani, I., Sundheim, B., Wilson, G. (2002) Instruction Manual for the Annotation of Temporal Expressions. MITRE Washington C3 Center, McLean, Virginia.

[61]. Fikes, R., Hayes, P., Horrocks, I. (2003) Owl ql: A language for deductive query answering on the semantic web. Technical Report 03- 14, Knowledge System Laboratory, Stanford University, Stanford, CA, 2003.

[62]. Fikes, R., Hayes, P. & Horrocks, I. (2004). OWL-QL-a language for deductive query answering on the Semantic Web. Web Semantics: Science, Services and Agents on the World Wide Web. 2 (1). p.pp. 19-29.

[63]. Fridman Noy, N., Musen, M.A. (1999). SMART: Automated Support for Ontology Merging and Alignment. In: Proceedings of the Twelfth Banff Workshop on Knowledge Acquisition, Modeling, and Management, Banff, Alberta.

[64]. Fridman, N., Musen, M.A. (2000). Prompt: Algorithm and tool for automated ontology merging and alignment. In Proc. of AAAI 2000.

[65]. García-Ramírez, F. “Una perspectiva en las pruebas de software en la Nube” Accesible at https://www.academia.edu/3646482/Una_Perspectiva_de_las_Pruebas_en_la_Nube .

[66]. GATE. A general architecture for text engineering. http://gate.ac.uk/ (Último acceso diciembre 2012)

[67]. Gartner Group 2013, http://www.gartner.com/newsroom/id/2573415

[68]. Goel, V., Byrne, W.J. (2000). "Minimum Bayes-risk automatic speech recognition". Computer Speech & Language 14 (2): 115–135. doi:10.1006/csla.2000.0138. Retrieved 2011-03-28.

[69]. Grau, B.C., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P. & Sattler, U. (2008). OWL 2: The next step for OWL. Web Semantics: Science, Services and Agents on the World Wide Web. 6 (4). p.pp. 309-322.

[70]. Green, T.R.G., Petre, M. (1996) Usability analysis of visual programming environments: A `cognitive dimensions' framework. Journal of Visual Languages and Computing, 7:131—174, 1996. [1]

[71]. Grimm, S., Motik, B., Preist, C. (2004). Variance in e-Business Service Discovery. Semantic Web Services Workshop. ISWC 2004.

[72]. Gruber, T. (2004). Interview Tom Gruber. SIGSEMIS Bulletin. 1 (3). p.pp. 4-9.

[73]. Gruber, T.R. (1993). A translation approach to portable ontology specifications. Knowledge Acquisition. 5 (2). p.pp. 199-220.

[74]. Guarino, N. (1995). Formal Ontology, Conceptual Analysis and Knowledge Representation. Int. J. Hum.-Comput. Stud. 43 (5-6). p.pp. 625-640.

[75]. Guarino, N. (1998). Formal Ontology and Information Systems. In: 1998, IOS Press, pp. 3-15.

[76]. Haarslev, V., Möller, R. (2001). RACER System Description. In Gor e, R., Leitsch, A., &Nipkow, T. (Eds.), Proc. of the 1st Int. Joint Conf. on Automated Reasoning (IJCAR 2001), Vol. 2083 of LNAI, pp. 701{706, Siena, Italy. Springer.

[77]. Haase, P. & Stojanovic, L. (2005). Consistent Evolution of OWL Ontologies. In: A. Gómez-Pérez & J. Euzenat (eds.). The Semantic Web: Research and Applications. Lecture Notes in Computer Science. [Online]. Springer Berlin Heidelberg, pp. 182-197. Available from: http://link.springer.com/chapter/10.1007/11431053_13. [Accessed: 30 May 2014].

[78]. Hahn, U., Schnattinger K. (1998a) Ontology engineering via text understanding. In IFIP'98 —Proceedings of the 15th World Computer Congress, Vienna and Budapest

[79]. Hameed, A., Sleeman, D., Preece, A. (2001). Detecting Mismatches Among Experts' Ontologies Acquired through Knowledge Elicitation. Research and Development in Intelligent Systems XVIII (Proceedings of ES 2001), BCS Conference Series, Springer-Verlag, 2001, pp 9-22.

[80]. Hastings, P. (1994). Automatic Acquisition of Word Meaning from Context. Ph.D. thesis, University of Michigan, Ann Arbor,MI.

[81]. Hearst, M.A. (1992) Automatic acquisition of hyponyms from large text corpora. In Proceedings of the 14th International Conference on Computational Linguistics. Nantes, France

[82]. Hearst, M.A., Hinrich, S. (1993). Customizing a lexicon to better suit a computational task. In Proc. Of the ACL-SIGLEXWorkshop on Acquisition of Lexical Knowledge from Text, Columbus, Ohio, USA, 1993

[83]. Herbst, N.R., Kounev, S., Reussner, R. (2013). “Elasticity in Cloud Computing: What It Is, and What It Is Not,” Present. Part 10th Int. Conf. Auton. Comput., pp. 23–27, 2013.

[84]. HermiT Reasoner. Recurso disponible on-line (Último acceso Feb.12): http://hermit-reasoner.com/2009/JAIR_benchmarks/

[85]. Hewlett-Packard Development Company. Hp lab semantic web research. Recurso disponible on-line (Último acceso Feb.12): http://www.hpl.hp.com/semweb.

[86]. Hewlett Packard. Jena SDB. Recurso disponible on-line (Último acceso Feb.12): http://openjena.org/wiki/SDB.

[87]. Hewlett Packard. Jena TDB. Recurso disponible on-line (Último acceso Feb.12): http://openjena.org/wiki/TDB.

[88]. Hewlett-Packard. Rdql: A query language for rdf. Re- curso disponible on-line (Último acceso Feb.12): http://www.w3.org/Submission/2004/ SUBM-RDQL-20040109, 2003.

[89]. Hobbs (1993) The generic information extraction system. In Proceedings of the Fifth Message Understanding Conference (MUC-5), Morgan Kaufmann

[90]. Högberg, D., Georgsson, E.F. (2012) “An Applied Evaluation and Assessment of Cloud Computing Platforms,” Jan. 2012.

[91]. Horrocks, I., Patel-Schneider, P.F. & Harmelen, F.V. (2003). From SHIQ and RDF to OWL: The Making of a Web Ontology Language. Journal of Web Semantics. 1. p.p. 2003.

[92]. HP. Jena. Recurso on-line (Último acceso Feb.12): http://jena. sourceforge.net/.

[93]. Huang, X., Acero, A., Hsiao-Wuen, H. (2007) Spoken Language Processing: A Guide to Theory, Algorithm and System Development 1st Edition.

[94]. Huang, X. (2001). Spoken Language Processing.

[95]. IEEE-SA Standard Board, Software Engineering Standards Comittee of the IEEE Computer Society. “IEEE Std 830-1998. IEEE Recommended Practice for Software Requirements Specifications”, *IEEE Standards Software Engineering, Vol. 4, Resource and Technique Standards*. The IEEE Inc., New York, 1999.

[96]. IETF. Rfc3987: Internationalized resource identifiers (iris). Recurso disponible on-line (Último acceso Feb.12):: http://www.ietf.org/rfc/rfc3987.txt.

[97]. Information Systems Group. Hermit owl reasoner. Recurso disponible on-line (Último acceso Feb.12): http://hermit-reasoner.com/.

[98]. ISO/IEC. Structured query language: Sql (iso/iec 9075 1:2008). Recurso disponible on-line (Último acceso Feb.12): http://www.iso.org/iso/catalogue_detail. htm?csnumber=45498.

[99]. ISO/IEC 29119 Pruebas de Software (Grupo de Trabajo de AENOR AEN/CTN71/SC7/GT26. http://in2test.lsi.uniovi.es/gt26/

[100]. Jackson, P., Schilder, F. (2006) "Natural Language Processing: Overview" in Encyclopedia of Language & Linguistics, ed. Keith Brown, Elsevier, Oxford, pp. 503-518.

[101]. James Hendler & Deborah L. McGuinness (2000). The DARPA Agent Markup Language. 16 (6). p.pp. 67-73.

[102]. Jelinek, F. (2001). Statistical Methods for Speech Recognition.

[103]. Joachims T. (1998) A probabilistic analysis of the Rocchio Algorithm with TFIDF for text categorization. Logic J. of the IGPL, 1998.

[104]. Jurafsky, Martin (2008) Speech and Language Processing.

[105]. Kaufman, L., Rousseeuw, P.J (1990) Finding Groups in Data: An Introduction to Cluster Analysis. John Wiley

[106]. Khan L., Luo F. (2002) Ontology Construction for Information Selection In proc. Of 14th IEEE International Conference on Tools with Artificial Intelligence, pp. 122-127, Washington DC, November 2002

[107]. Kietz, J., Maedche, A., Volz, R. (2000) A Method for Semi-Automatic Ontology Acquisition from a Corporate Intranet. Workshop on Ontologies and Texts. Juan-les-Pins

[108]. King, C. (2009). Advanced BlackBerry Development (1st ed.). Apress. p. 350. ISBN 1430226560.

[109]. Kit, E. (1995) Software Testing in the Real World. ACM Press.

[110]. Klein, M. & Fensel, D. (2001). Ontology versioning on the Semantic Web. In: Stanford University. 2001, pp. 75-91.

[111]. Klyne, G., Carroll, J.J. (2004) RDF Concepts and Abstract Syntax. s.l. : W3C.

[112]. Kongthon, A., Sangkeettrakarn, C., Kongyoung, S., Haruechaiyasak, C. (2009) Implementing an online help desk system based on conversational agent Authors.

Published by ACM 2009 Article, Bibliometrics Data Bibliometrics. Published in: Proceeding, MEDES '09 Proceedings of the International Conference on Management of Emergent Digital EcoSystems, ACM New York, NY, USA. ISBN 978-1-60558-829-2, doi:10.1145/1643823.1643908

[113]. Lehmann, Fritz. Semantic networks. Computers Math. Applic., vol 23, No. 2-5, pp. 1-50, 1992.

[114]. Lenat, D. (1995). Cyc: A large-scale investment in knowledge infrastructure. Communications of the ACM, 38(11):33–38, Nov.

[115]. Lesk, M. (1986). Automated sense disambiguation using machine-readable dictionaries: How to tell a pine cone from an ice cream cone, en Proceedings of the 1986 SIGDOC Conference, Association for Computing Machinery, págs. 24–26, Toronto, Canada.

[116]. Lisa Inference Engine. http://lisa.sourceforge.net/

[117]. LLISTERRI, J. (2003) "Las tecnologías del habla para el español", Seminario Ciencia, Tecnología y Lengua Española: La terminología científica en españo, vol. 11

[118]. LLISTERRI, J. ()2003), "Lingüística y tecnologías del lenguaje", Lynx.Panorámica de Estudios Lingüísticos, vol. 2, pp. 9-71.

[119]. Luke, S., Spector, L., Rager, D. & Hendler, J. (1997). Ontology-based Web Agents. In: Proceedings of the First International Conference on Autonomous Agents. AGENTS '97. [Online]. 1997, New York, NY, USA: ACM, pp. 59-66. Available from: http://doi.acm.org/10.1145/267658.267668. [Accessed: 31 March 2014].

[120]. Madhavan, J., Bernstein, P., Rahm, E. (2001) Generic schema matching with cupid. In Proceedings of the 27th VLDB Conference, Rome, Italy.

[121]. Maedche, A., Staab, S. (2001). Ontology learning for the Semantic Web. IEEE Intelligent Systems, 16(2), 72-79.

[122]. Maedche, A., Staab, S. (2004) "Ontology learning". En: Staab, S.; Studer, R. (eds.). Handbook on ontologies. Berlin: Springer

[123]. Malhotra, A. (2009) Incubator group report. Recurso disponible on-line: http://www.w3.org/2005/Incubator/rdb2rdf/XGR/, 2009.

[124]. Manning, C.D., Schuetze, H. (1999) Foundations of Statistical Natural Language Processing. MIT Press, Cambridge, Massachusetts

[125]. Marcos-Marín, F.A. (2000) "Introducción a la gramática" in Introducción a la lingüística española, ed. M. AlvarAriel, Barcelona, pp. 23-49

[126]. McDermott, D.V., Dou, D. (2002) Representing disjunction and quantifiers in rdf. In Proceedings of International Semantic Web Conference, pages [250,263], 2002.

[127]. McGuinness, D.L. & Van Harmelen, F. (2004). OWL web ontology language overview. W3C recommendation. 10 (10). p.p. 2004.

[128]. Millar, A. (1990). WordNet: An On-line Lexical Resource. Journal of Lexicography, 3(4).

[129]. Miller, G.A. (1995). Wordnet: A lexical database for English. Commun. ACM, 38(11), 39–41.

[130]. Miller, L., Seaborne, A., Reggiori, A. (2002). Three implementations of squishql, a simple rdf query language. In International Semantic Web Conference, 2002.

[131]. Minsky, M., (1975) A framework for representing knowledge, P. H. Winston (ed.), The Psychology of Computer Vision, McGraw-Hill, 1975.

[132]. Mizoguchi, R. (2004). Ontology Engineering Enviroments, in S. Staab and R. Studer, editors, Handbook on Ontologies, pages 173-189. Springer, 2004

[133]. Mohri, M. (2002). "Edit-Distance of Weighted Automata: General Definitions and Algorithms". 957–982.doi:10.1142/S0129054103002114. Retrieved 2011-03-28.

[134]. Morin. E. (1999) Automatic acquisition of semantic relations between terms from technical corpora. In Proc. Of the Fifth International Congress on Terminology and Knowledge Engineering - TKE'99

[135]. Navigli R., Velardi P. and Gangemi A. (2003). Ontology Learning and Its Application to Automated Terminology Translation. IEEE Intelligent Systems, January/February 2003, 22-31

[136]. Navigli R. and Velardi P. (2004). Learning Domain Ontologies from Document Warehouses and Dedicated Web Sites. Computational Linguistics, vol 30(2), 151-179

[137]. Omelayenko, B. (2001). Learning of ontologies for the Web: The analysis of existent approaches. Proceedings of the International Workshop on Web Dynamics.

[138]. Open RDF. Sesame. Recurso disponible on-line (Último acceso Feb.12): http://www. openrdf.org/.

[139]. Openlink software. Vistas rdf de virtuoso. Recurso disponible on- line (Último acceso Feb.12): http://virtuoso.openlinksw.com/whitepapers/relational% 20rdf%20views%20mapping.html.

[140]. Openlink software. Servidor universal openlink virtuoso. Recurso disponible on-line (Último acceso Feb.12): http://virtuoso.openlinksw.com/.

[141]. Parsia, B., Sirin, E. (2004). Pellet: An OWL-DL Reasoner. Poster at the 3rd Int. Semantic Web Conference (ISWC 2004).

[142]. Peter, M., Grance, T. (2009) "The NIST definition of cloud computing." National Institute of Standards and Technology 53.6 (2009): 50.

[143]. Pereira, F., Tishby, N., Lee, L. (1993) Distributation Clustering of English Words. In Proceedings of the ACL-93, 1993, pages 183–199

[144]. Pierce, J. (1969). "Whither Speech Recognition". Journal of the Acoustical Society of America.

[145]. Pieraccini, R. (2012). The Voice in the Machine. Building Computers That Understand Speech.

[146]. Poli, R. (2003). Descriptive, Formal and Formalized Ontologies. In: Husserl's Logical Investigations Reconsidered. Contributions to Phenomenology. [Online]. Springer Netherlands, pp. 183-210. Available from: http://link.springer.com/chapter/10.1007/978-94-017-0207-2_12. [Accessed: 29 May 2014].

[147]. Pressman, R.S. Ingenieria de Software Un Enfoque Práctico

[148]. Protégé. Recurso disponible on-line (Último acceso Feb.12): http://protege.stanford.edu/

[149]. Prud'Hommeaux, E. & Seaborne, A. (2008). SPARQL query language for RDF. W3C recommendation. 15.

[150]. Quinlan, J. R. C. Programs for Machine Learning. San Mateo Morgan Kaufmann.

[151]. Rabiner, L. (1993). Fundamentals of Speech Recognition.

[152]. Racer Systems GmbH & Co. KG. Racer pro. Recurso disponible on-line (Último acceso Feb.12): http://www.racer-systems.com.

[153]. Rahm, E., Bernstein, P. (2001) A survey of approaches to automatic schema matching. The VLDB Journal, 10:[334,450].

[154]. Reeve L., Han H., (2005) Survey of semantic annotation platforms. In SAC ’05: Proceedings of the 2005 ACM symposium on Applied computing, pages 1634–1638, New York, NY, USA, 2005. ACM Press.

[155]. Resnik P., Smith N.A. (2003) The Web as a Pararell Corpus. Computational Linguistics. Vol 29(3), 349-380

[156]. Rhoton, J. (2010). Cloud Computing Explained. 2nd edition. Recursive Press, Tunbridge Wells.

[157]. Richard Cyganiak, David Wood & Markus Lanthaler (2004). RDF Vocabulary Description Language 1.0: RDF Schema. Changes. [Online]. Available from: http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/. [Accessed: 30 May 2014].

[158]. Rizk, A. (2009). Beginning BlackBerry Development (1st ed.). Apress. p. 264. ISBN 1430272252.

[159]. Sahoo, S.S., Halb, W., Hellmann, S., Idehen, K., Thibodeau, I., Auer, S., Sequeda, J., Ezzat, A. (2012). A survey of current approaches for mapping of relational databases to rdf. Recurso disponible on-line (Último acceso Feb.12): http://www.w3.org/2005/Incubator/rdb2rdf/RDB2RDF_SurveyReport.pdf.

[160]. Salas-Zárate, M., Colombo-Mendoza, L. (2013) “Cloud Computing; A review of PAAS, IAAS SAAS Services and providers”. Revista Digital de la Facultad de Ingenierías" Lámpsakos", 1.7 (2013), 47-57.

[161]. Saquete Boró, E. (2005). Resolución de información temporal y su aplicación a la búsqueda de respuestas. Tesis doctoral. Universidad de Alicante.

[162]. Schroeder, M. (2004) Computer Speech.

[163]. Schubert, L., Jeffery, K. & Neidecker-Lutz, B. (2012). Advanced Cloud Technologies under H2020.

[164]. Seaborne, A. (2004). RDQL-a query language for RDF. W3C Member submission. 9 (29-21). p.p. 33.

[165]. Staab, S., Schnurr, H.P., Studer, R., Sure, Y. (2001) Knowledge Processes and Ontologies, IEEE Intelligent Systems, 16(1).

[166]. Staab, S., Studer, R., 2004. Handbook on Ontologies (International Handbooks on Information Systems). 1st Edn., Springer, New York, ISBN-13: 978-3540408345.

[167]. Suominen, H., Zhou, L., Hanlen, L., Ferraro, G. (2015). Benchmarking Clinical Speech Recognition and Information Extraction: New Data, Methods, and Evaluations. JMIR Medical Informatics, 3(2), e19.http://medinform.jmir.org/2015/2/e19/

[168]. Suryanto, H., Compton, P. (2000) Learning Classification taxonomies from a classification knowledge based system. In: S. Staab, A. Maedche, C. Nedellec, P. Wiemer-Hastings (eds.), Proceedings of the Workshop on Ontology Learning, 14th European Conference on Artificial Intelligence ECAI'00, Berlin, Germany, August 20-25, 2000

[169]. TimeML. Markup Language for Temporal and Event Expressions. http://www.timeml.org/site/index.html (Último acceso diciembre 2012).

[170]. TimeML Specifications. http://www.timeml.org/site/publications/specs.html.

[171]. TimeML Documentation. http://www.timeml.org/site/publications/timeMLdocs/timeml_1.2.1.html.

[172]. Tsarkov, D., Horrocks, I. (2004) Reasoner Demonstrator. s.l. : Ontology Infraestructure for Semantic Web.

[173]. Tsarkov, D., Horrocks, I. (2006). FaCT++ Description Logic Reasoner: System Description. In Proc. of the 3rd Int. Joint Conf. on Automated Reasoning (IJCAR 2006), Vol. 4130 of LNAI, pp. 292{297, Seattle, WA, USA. Springer.

[174]. Tsarkov, D. Fact++. Recurso disponible on-line (Último acceso Feb.12): http://owlman.ac.uk/factplusplus/.

[175]. Universidad Politécnica de Valencia. Las plataformas para móviles en 2013 @UPV. Accesible en http://www.youtube.com/watch?v=N6NlbpN1zVk

[176]. Valencia-García, R., Ruiz-Sánchez, JM., Vivancos-Vicente, P.J., Fernández-Breis, J.T. and Martínez-Béjar, R. (2004) An incremental approach for discovering medical knowledge from texts. Expert Systems with applications, VOL. 26, 291--299, (2004)

[177]. Valencia-García, R (2005). Un entorno para la extracción incremental de conocimiento desde texto en lenguaje natural. Tesis doctoral. ISSN: 84-689-5046-7

[178]. Valencia-García, R., Castellanos-Nieves, D., Fernández-Breis, J.T. and Vivancos-Vicente, P.J. (2006). A methodology for extracting ontological knowledge from Spanish documents. Lecture Notes in Computer Science, VOL. 3878, 71—80

[179]. Vázquez Pérez, S. (2009). Resolución de la ambigüedad semántica mediante métodos basados en conocimiento y su aportación a tareas de PLN Tesis doctoral. Universidad de Alicante

[180]. Veen, J. "Case Study: Intranets, Usability, and Value". "All too often an intranet redesign fails because users don't adopt the new mechanisms for creating and distributing content — this can be mitigated by choosing early projects that focus on commonly recognized quick wins. Her plan should have a long-term vision, but it should be implemented in small increments that instill a sense of confidence and value among users throughout the redesign. Small increments allow for course corrections toward the long-term vision."

[181]. Volz, R., Oberle, D., Staab, S., Studer, R. (2002) Ontolift prototype. wonderweb project. Technical report, Technical Report D11, 2002.

[182]. Vrandecic D., Sofía-Pinto, S., Sure, Y. (2005) Journal of Knowledge Management. The DILIGENT Knowledge Processes, 9 85-96

[183]. W3C. Semantic Web Architecture. http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html

[184]. W3C (2012a). Owl 2 functional syntax. [Online]. Available from: http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/. [Accessed: 30 May 2014].

[185]. W3C (2012b). Owl 2 xml serialization. [Online]. Available from: http://www.w3.org/TR/owl2-xml-serialization/. [Accessed: 30 May 2014].

[186]. W3C (2014). Owl 2 rdf/xml syntax specification. [Online]. Available from: http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/. [Accessed: 30 May 2014].

[187]. W3C. Sparql protocol for rdf. Recurso on-line (Último acceso Feb.12): http://www.w3.org/TR/rdf-sparql-protocol/.

[188]. W3C. Sparql query results xmlf ormat. Recurso on-line (Último acceso Feb.12): http://www.w3.org/TR/rdf-sparql-XMLres/.

[189]. W3C. World wide web consortium. Recurso on-line (Último acceso Feb.12): www.w3c.org.

[190]. W3C. Rdb2rdf working group. Recurso on-line (Último acceso Feb.12): http://www.w3.org/2001/sw/rdb2rdf/, 2009.

[191]. W3C. A direct mapping of relational data to rdf. Recurso on-line (Último acceso Feb.12): http://www.w3.org/TR/rdb-direct-mapping/.

[192]. W3C. Lenguaje de consultas sparql. Recurso on-line (Último acceso Feb.12): http://www.w3.org/TR/rdf-sparql-query.

[193]. W3C. R2rml: Rdb to rdf mapping language. Recurso on-line (Último acceso Feb.12): http://www.w3.org/TR/r2rml/

[194]. Warschauer, M., Healey, D. (1998). Computers and language learning: An overview. Language Teaching, 31, 57-71.

[195]. Webb, G., Wells, J., Zheng, Z. (1999) An Experimental Evaluation of Integrating Machine Learning with Knowledge Acquisition. Machine Learning, 31(1): 5-23

[196]. Wilbur Semantic Web Toolkit for CLOS. http://wilbur-rdf.sourceforge.net/

[197]. Winograd, T. (1972). Understanding natural language. Cognitive Psychology. 3 (1). p.pp. 1-191.

[198]. Yu, D., Deng, L. (20014) Automatic Speech Recognition: A Deep Learning Approach. Publisher: Springer.