UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE INGENIERÍA ELÉCTRICA y ELECTRÓNICA

"IMPLEMENTACION DE UNA APLICACION WEB PARA INDIZAR RECURSOS DE INFORMACION CON TERMINOS RECUPERADOS DE UN TESAURO USANDO UN WEBSERVICE"

TESIS

PARA OPTAR EL GRADO ACADÉMICO DE MAESTRO EN CIENCIAS CON MENCIÓN EN TELEMÁTICA

ELABORADO POR

ADAM HEGEL SÁNCHEZ AYTE

ASESOR

M. Sc. ARTURO VILCA ROMÁN

LIMA - PERÚ

2012

A mi esposa Rosa,
a mis padres Ana y Feliciano,
a mis hermanos Javier, Jorge, Diana y Ricardo
y a los amigos de toda la vida.

Agradezco a mi asesor,
Msc. Arturo Vilca por su invalorable ayuda.
A Johannes Keizer, Valeria Pesce,
Imma Subirats y Ahsan Morshed por
estimularme a profundizar en estos temas.
A Antonella Picarella, Giampaolo Rugo,
Erna Klupacs y Teresa Iniesta
por su invalorable amistad en Roma.
A Miguel Saravia, amigo y maestro,
por iniciarme en el tema hace muchos años
y a mi querido Luis Fernando Crespo
por confiar siempre en mí.

ÍNDICE GENERAL

LISTA DE FIGURAS

LISTA DE TABLAS

RESUMEN

Se describe el desarrollo de una aplicación web tipo cliente-servidor que automatiza la selección de términos que identifican temáticamente un recurso de información dentro de su proceso de registro bibliográfico.

Los términos temáticos son recuperados de un sistema de organizacion de conocimiento denominado AGROVOC. AGROVOC es un tesauro que contiene una base de datos de términos o expresiones para diferentes lenguajes relacionados con agricultura, silvicultura, pesca y dominios relacionados. AGROVOC está auspiciado por la FAO (Food and Agriculture Organization), una entidad que forma parte de las UN (United Nations).

La aplicación web permite la asignación de descriptores temáticos a un recurso en dos formas: por búsqueda incremental ó por extracción automática de términos del archivo catalogado en base a un algoritmo. Una vez seleccionados los términos temáticos que describen el recurso de información, se almacenan en todos los idiomas disponibles ofrecidos por el catálogo bibliográfico online junto con la URI (Uniform Resource Identifier) que le corresponde según la FAO de forma que puedan ser referenciados en la web semántica.

ABSTRACT

The thesis describes a client-server software application for facilitating or automating the process of selecting keywords for the subject-indexing of bibliographic resources.

The terms identifying the subjects are retrieved from a Knowledge Organization System named AGROVOC. AGROVOC is a thesaurus of around 40,000 terms related to agriculture, fisheries, forestry and related domains. AGROVOC is sponsored by FAO (the Food and Agriculture Organization of the United Nations), a specialized agency of the UN (United Nations).

The web application allows to assign thematic descriptors to a resource in either of the following modes: by searching for words and getting suggestions of relevant terms or by letting the application automatically extract terms from the full text attached to the resource, based on an algorithm.

Once terms are selected in either way, they are used to index the resources and their labels are stored in all the languages enabled in the website along with their semantic web identifiers (URIs).

INTRODUCCIÓN

El presente trabajo describe el desarrollo de una herramienta para automatizar el trabajo de indización temática de recursos de información relacionados con agricultura. La herramienta fue implementada bajo licencia GNU y se ejecuta bajo el sistema de gestión de contenidos denominado Drupal.

El tema fue elegido debido a la necesidad de contar con herramientas que faciliten la catalogación de un objeto de información de acuerdo a estándares y disminuyan la tasa de errores en que normalmente se incurre. Esta mejora en la calidad de los metadatos que se usan para catalogar un recurso de información garantizarán su recolección desde otras redes de información, en especial, de “redes inteligentes” como la web semántica.

La idea de construir una red global de información inteligente está basada en la forma como asociamos información en nuestros cerebros para producir conocimiento. Si bien el estado de la ciencia actual ha demostrado que la asociación es libre y basada en patrones, los 20 años de vida de la web están demostrando que la única forma que las máquinas emulen un comportamiento similar es contando con enormes bases de datos con recursos catalogados con la mayor precisión posible y en plena coherencia con estándares libres que garanticen su independencia, interoperatibilidad y preservación.

La metodología para el desarrollo de la aplicación de software es del tipo cascada. Actualmente se está usando el proyecto Global Rangelands de la universidad de Arizona en USA y como parte de la distribución AgriDrupal promovida por la FAO.

CAPÍTULO 1

ANTECEDENTES Y PLANTEAMIENTO DEL PROBLEMA

1.1. Antecedentes

Después de la segunda guerra mundial se produce un gran interés en el desarrollo de nuevas técnicas de indización y búsqueda de información debido al crecimiento exponencial de la producción de conocimiento científico especializado. Como los tradicionales esquemas de clasificación temática se habían convertido en muy generales y rígidos para representar los recursos de información, los científicos elaboraron conceptos importantes como la indización (subject indexing), la clasificación temática post-coordinada, la elección de vocabularios para representar los descriptores temáticos de un documento y el desarrollo de modelos de recuperación de información basadas en estadísticas.

La aparición de la computadora con su poderosa capacidad de procesamiento fue clave para el desarrollo de tecnologías basada en estos conceptos. Este es el caso de la indización que se definió como el proceso de descripción o clasificación de recursos de información a través de términos temáticos (también llamados metadatos basados en estándares de valores). Los términos pueden ser seleccionados directamente del texto, por etiquetado libre (free tagging) ó de sistemas de organización de conocimientos como los tesauros. Entre los finales de los años cincuenta y los inicios de los años sesenta se realizaron muchas investigaciones para automatizar los procesos de indización basadas en análisis estadístico. Estas técnicas trabajaban con textos alfanuméricos tratando de extraer de ellos, la frecuencia de vocablos, partes de palabras o frases así como su posición. Entre los trabajos desarrollados en esa época se pueden citar: las técnicas de clasificación formal para vocabularios de texto desarrolladas por la Cambridge Language Research Unit y y el inicio de la investigación de Gerard Salton alrededor del proyecto System for the Mechanical Analysis and Retrieval of Text (SMART) en la universidad de Cornell.

El apogeo de la Inteligencia Artificial (IA) durante los años setenta aplicaría su enfoque simbólico a la comprensión del lenguaje natural que luego derivaría en un análisis sintáctico en términos de formalismos y de algoritmos de análisis. Sin embargo, a pesar de las grandes expectativas que se tenían, la IA nunca demostraría ser superior a los enfoques estadísticos aplicados a casos similares.

En la década de los ochenta, muchos experimentos mostraron que las ideas iniciales acerca del valor del análisis estadístico en la recuperación de información estaban justificadas aunque su uso había sido ínfimo en sistemas fuera de los am-

bientes académicos. Por otro lado, durante está época se desplazó la atención en la implementación de interfases de búsqueda humano/máquina para especialistas hacia interfases para usuarios finales sin ninguna preparación previa. La revolución informática había llegado y las computadoras se estaban orientando al usuario final, el internet comenzaba a proveer acceso remoto a toda clase de archivos y los procesadores de texto empezaban a masificarse.

La década de los noventa marca el resurgimiento de la recuperación de información basada en análisis estadístico en medio de los esfuerzos de la Text REtrieval Conferences (TRECs) y la web por posicionarla fuera de los laboratorios de investigación. Los motores de búsqueda web, ajenos a los compromisos y supuestos de los servicios bibliográficos, terminarían usando con mayor libertad estas técnicas estadísticas, como fue el caso de Altavista y luego Google, aunque en una forma diferente. También se inician los estudios en IA, teoría de ondas y morfología matemática para una interpretación automática del contenido de imágenes, gráficos y sonido. Por otro lado, las TRECs durante servirían para presentar el Modelamiento del Lenguage (Language Modelling) como una nueva forma de análisis probabilístico.

Durante la primera década del siglo XXI, una serie de reuniones con expertos y socios internacionales motivaron a la Food and Agriculture Organization (FAO) a liderar junto con sus contrapartes en cada país un proceso que establezca un nuevo modelo de gestión de información agrícola descentralizada basada en intercambio de datos interoperables a través de la web. Esta iniciativa estuvo inspirada en la conferencia XML2000, donde Tim Berners Lee, el inventor de la web, presentó su visión de la web semántica basada en ontologías.

Como resultado de esta decisión, la FAO a través de su oficina Agricultural Information Management Standards (AIMS) se comprometió a:

a) Usar metadata descriptiva simple para integrar y facilitar el acceso a información agrícola entre paises desarrollados y países en vías de desarrollo.

b) Desarrollar y mantener tesauros y ontologías como fuentes de descriptores que faciliten el acceso estructurado a información agrícola.

c) Fortalecer redes, desarrollar capacidades y promover la divulgación de estas normas a través de sus contrapartes y socios.

La FAO ya tenía mucha experiencia en el trabajo con informacion agraria. Desde la década de los setenta ya contaba con International System for Agricultural Scien-

ce and Technology (AGRIS), una base de datos con 2.6 millones de registros bibliográficos y con un tesauro denominado AGROVOC para indizarlo temáticamente. AGROVOC había sido iniciado en 1980 y desde entonces se define como un tesauro multilingue, estructurado y controlado cuyos términos abarcan la terminología de todos los ámbitos de la agricultura, la silvicultura, la pesca y las áreas relacionadas con los alimentos (como el medio ambiente). Tiene como objetivo estandarizar la descripción semántica de los recursos de información facilitando la indexación, recuperación y organización de datos en sistemas de información agrícola. Es usado en todo el mundo por investigadores, bibliotecarios, gestores de información y está disponible en los seis idioma oficiales de la FAO: inglés, fránces, español, árabe, chino y ruso.

Hacia finales de la década pasada, la FAO empieza a trabajar en conjunto con sus contrapartes nacionales y aliados técnicos para el desarrollo de herramientas que faciliten el uso de AGROVOC en los procesos de indización manual y automática de bases de datos bibliográficas, almacenando además el URI que le corresponde a cada término utilizado en la indización. De este modo el recurso indizado puede formar parte de la web semántica al estar disponible para cualquier consulta SPARQL (Simple Protocol and RDF Query Language) que incluya los URIs de los índices que lo describen temáticamente.

Entre las herramientas desarrolladas a la fecha están un demo para la indexación automática denominada AgroTagger que usa el algoritmo KEA (Keyphrase Extraction Algorithm) para la extracción de términos de los documentos y un buscador de términos AGROVOC disponible en la página principal de AIMS.

1.2. Justificación

La indización de recursos de información bibliográfica es clave para facilitar su posterior recuperación (*findability*) y siempre estuvo reservada a especialistas que se encargaban de identificar el contenido temático del documento de acuerdo a su experiencia y sus conocimientos. Durante este proceso, estos profesionales debían decidir la exhaustividad (número) y la especificidad de los términos para definir la profundidad de la indización. Asimismo, los términos podían ser elegidos directamente del texto ó desde un vocabulario controlado como un tesauro. Por otro lado, muchas veces era importante la secuencia en que los términos eran presentados (indización pre-coordinada).

En vista que el proceso clásico de indización como el que hemos descrito queda desbordado ante las toneladas de información bibliográfica en todos los idiomas

que se tienen que manejar actualmente en la web, se necesita automatizar partes o todo su proceso con la finalidad de indizar más rápido sin perder exhaustividad y especifidad en los términos.

Con ese fin se hace necesario la implementación de una herramienta que automatice la selección y la asignación de los términos de un vocabulario controlado cuando se indiza un recurso en varios idiomas. Además dada la progresiva conversión de muchas bases de datos como la Wikipedia en bases de datos semánticas a fin de cruzar información entre todas ellas y convertir a la web en una enorme base de datos global, se hace necesario almacenar también el URI que identifica a cada término procedente del vocabulario controlado con la finalidad que el documento indizado pueda ser recuperado también en este medio.

Se ha escogido el tesauro AGROVOC como vocabulario controlado para la indización debido a que tiene 580 000 términos (todos con su URI respectiva y publicada) relacionados con agricultura, la silvicultura, la pesca y las áreas relacionadas con los alimentos, además de estar disponible en 20 lenguajes. También se hace necesario la elección de una plataforma o framework bajo la cual desarrollar la aplicación. Se ha elegido Drupal, un framework escrito en lenguage PHP para el desarrollo de sistema de gestión de contenidos, debido a que cuenta con un API (*Application Programming Interface*) muy robusta y bien documentada además de una comunidad de usuarios y desarrolladores que garantiza la sostenibilidad de la aplicación a largo plazo.

En conclusión, se desarrollará una aplicación web sobre Drupal que permitirá seleccionar automáticamente términos del tesauro AGROVOC mediante webservices al momento de la indización de un recurso de información. Además almacenará las correspondientes versiones idiomáticas de cada término en la base de datos junto con su URI.

1.3. Objetivos y Alcances de la tesis

1.3.1. Objetivo General

Implementar una aplicación web para indizar automáticamente recursos de información con términos recuperados del tesauro AGROVOC mediante un webservice

1.3.2. Objetivos Específicos

1. Desarrollar una aplicación web (widget) para automatizar la selección de términos mediante búsqueda incremental.

2. Desarrollar una aplicación web (widget) para automatizar la selección de términos a través de su extracción automática de un archivo digital.

1.4. Organización de la tesis

El capítulo I describe los antecedentes, justificación, objetivos, alcances y organización de la tesis.

El capítulo II expone los fundamentos básicos de los metadatos, reseña los estándares más importantes y explica los procesos de indización manual y automática. También se desarrolla en esta parte los conceptos más importantes en torno al algoritmo *Keyphrase Extraction Algorithm* (KEA) para la extracción automática de términos de un archivo.

El capítulo III explica de manera general los conceptos claves de la web semántica: la infraestructura de descripción de recursos (RDF), el esquema RDFS, el Lenguaje de Ontologías Web (OWL) y el concepto *Linked Data*. Esta parte es clave para entender la importancia de los URIs que genera nuestra aplicación web por cada término que indiza.

El capítulo IV presenta de manera concisa todo los detalles relacionados con la arquitectura de Drupal, el sistema que nos servirá para construir nuestra aplicación.

El capítulo V describe el análisis de requerimientos funcionales y no funcionales, así como el diseño, programación y pruebas de la aplicación.

El capítulo VI expone las conclusiones y recomendaciones de la tesis.

CAPÍTULO 2
FUNDAMENTOS DE METADATOS

2.1. Conceptos Básicos

El término metadata fue usado antes de la aparición del internet, principalmente en ciencias de la computación para especificar información sobre bases de datos y programas. Aunque literalmente se entiende como “datos sobre datos” ó “información sobre información” se define como “información estructurada que describe, explica, localiza o dicho de otro modo facilita la recuperación, uso y gestión de un recurso de información” [1] .

En este período la creación y la gestión de los metadatos fue responsabilidad exclusiva de los profesionales de la información y la organización de la información en bibliotecas, archivos y museos e instituciones similares estaba normada por estándares como la Anglo-American Cataloguing Rules Revised (AACR2) y MARC (Machine-Readable Cataloging), mientras que la representación de los contenidos era normada por esquemas de clasificación y listas de encabezados de materia como la Dewey Decimal Classification (DDC) y la Library of Congress Subject Headings (LCSH).

Los metadatos son muy importantes porque facilitan el acceso al recurso no sólo dentro de la base de datos local sino desde una base de datos remota a través de búsquedas federadas (harvesting) que usan el protocolo OAI-PMH ó a través de metabúsquedas (metasearching) que usan el protocolo Z39.50. También son relevantes porque permiten mantener los contextos de los recursos y sus complejas interrelaciones entre ellos mismos además de sus asociaciones con personas, lugares y eventos. Tal es el caso de las versiones digitales de libros físicos cuya metadata debería registrar el vínculo que existe entre ambos además de los vínculos que pudiera tener con otras versiones digitales del mismo. Por otro lado, los metadatos garantizan de alguna forma la preservación y la persistencia de los recursos al documentar cómo fueron creados, mantenidos y cómo funcionaban. Esto es clave por ejemplo para que los recursos multimedia tengan la oportunidad de sobrevivir migraciones a través de sucesivas generaciones de hardware y software.

La aparición de internet hizo que los metadatos salieran de los ambientes académicos y se abrieran al público en general. Sin embargo, debido a la complejidad de su uso, aparecieron otros más simples que pronto se hicieron muy populares: metatags, palabras claves (folksonomías) y marcadores sociales (social bookmarks). Cualquiera que haya subido un video a Youtube ó una foto a Flickr ha usado folk-

sonomías para registrar información sobre su recurso. Cualquiera que haya usado Delicious para guardar la dirección de un sitio web ha usado folksonomías para registrar información sobre su recurso y compartir su marcador (bookmark) en su red social.

El contenedor de los metadatos puede ser digital ó análogico. Por mucho tiempo la comunidad bibliotecaria han estado creando metadatos en formatos analógicos como catálogos de tarjetas, archivos verticales y etiquetas de archivos mucho antes de ser incorporados a los sistemas de información digital. Por otro lado, los metadatos no sólo se encargan de describir un objeto sino también de registrar información sobre su contexto, gestión, procesamiento y preservación. Asimismo, los metadatos no sólo pueden ser generados por humanos sino también por algoritmos computacionales o inferidos de una relación lógica con otro recurso.

La siguiente figura ilustra las diferentes fases que sigue la vida digital de un recurso de información. Conforme se va moviendo entre sus diferentes fases, el recurso va incorporando más capas de metadatos siguiendo el mismo proceso de la bola de nieve. Estas capas pueden incorporarse al recurso de manera manual o automática. Pueden incorporarse en la misma envoltura del recurso (codificado en la cabecera del archivo) o referenciarlo de manera externa (si se le registra en un nuevo nivel de agregación).

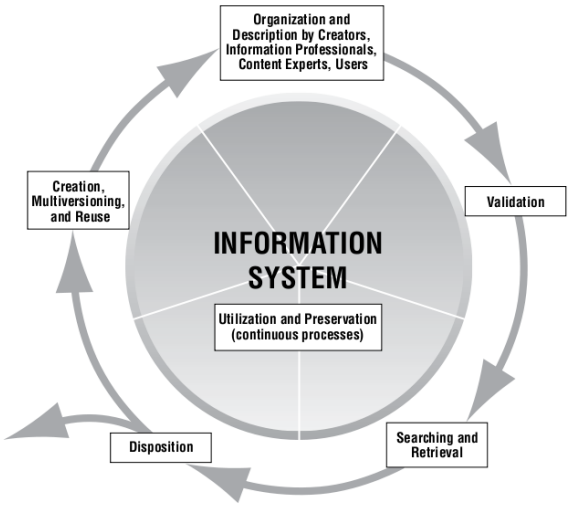

Figura 2.1: Ciclo de Vida de un Recurso Digital. Fuente: [1]

2.2. Estándares de metadatos para registro de recursos de información

2.2.1. Estándares para estructuras de datos

Son metadatos que norman la estructura (esquema) que debe seguir la descripción de un recurso de información. Su implementación ayuda a: catalogar recursos, implementar ayudas de búsquedas, diferenciar versiones, implementar índices especializados, establecer relaciones entre recursos y gestionar anotaciones de autores y usuarios.

Entre los más importantes tenemos:

a) MARC (*Machine-Readable Cataloging format*).

Es un estándar que tiene varios formatos para diferentes tipos de datos y es ampliamente usado para registrar y compartir libros y materiales afines. Fue desarrollado en la década de los sesenta por Henriette Avram para la Biblioteca del Congreso de los Estados Unidos cuando empezaron a usar computadoras. La primera versión fue denominada LC MARC. La siguiente versión fue MARC 21 y se ha convertido en el estándar usado por muchas bibliotecas en el mundo.

El formato MARC define tres aspectos: la estructura del registro basada en el estándar ISO 2709, la especificación de los campos dentro de cada registro basada en números de 3 dígitos y el contenido del registro. Para este último aspecto se apoya en los estándares AACR (Anglo-American Cataloguing Rules) y LCSH (Library of Congress Subject Headings) para la provisión de términos autorizados.

Los formatos MARC abarcan los siguientes tipos: registros de autoridad (authority files, registro bibliográfico (Bibliographic records), registro de clasificación (Classification records), registros de servicio público (Community Information records) y registro de localización de copias (Holdings records).

b) DCMES (*Dublin Core Metadata Element*)

Es un estándar para registrar documentos publicados en la web. Surgió en 1995 en la Metadata Workshop realizada en Dublín (Ohio, USA) frente a la proliferación de recursos electrónicos y ante la imposibilidad de que los bibliotecólogos puedan cubrirlos todos. Era un tiempo en que coexistían dos formas de describir recursos en web: por un lado, los motores de búsqueda que indexaban contenidos pero con pocos metadatos y por el otro lado, los catálogos electrónicos construídos en formato MARC pero muy costosos en su implementación para ser adoptado por la comunidad web. En vista de esta situación, el workshop decidió plantear un esquema que

supere estos dos extremos, el resultado fue la definición de 13 elementos y algunas reglas básicas que pudieran ser aplicadas por las personas sin conocimientos especializados.

Al poco tiempo, los 13 elementos fueron incrementados a 15 tal como se puede ver en la tabla siguiente:

Categorías	**Elemento**
Content	Title Description Type Subject Source Relation Coverage
Intellectual Property	Creator Publisher Rights Contributor
Instantiation	Date Format Identifier Language

Cuadro 2.1: Elementos Dublin Core. [16]

Finalmente, esta tensión llevó a la extensión del estándar de manera que permita refinar la semántica en algunos campos. Esta versión fue lanzada el 2000 y fue llamada *Qualified Dublin Core*. La estructura actual se puede ver en la tabla 2.2.

2.2.2. Estándares para valores de datos

Son metadatos que norman los valores que se tienen que llenar cuando se registra un contenido de información. Entre los más importantes tenemos

a) LCSH (Library of Congress Subject Headings)

Es un vocabulario controlado de encabezados de materia mantenido por la Biblioteca del Congreso de los Estados Unidos desde 1898 para su uso en el registro bibliográfico. Tiene como propósito proporcionar al usuario puntos de acceso temático.

Elementos	Refinamiento	Esquema de Codificación
Title	alternative	
creator		
subject		DDC, LCC, LCSH, NLM, UDC, MESH
description	tableOf Contents, abstract	
publisher		
contributor		
date hline type	created, valid, issued, modified, dateCopyrighted, dateSubmitted, available, dateAccepted	W3CDTF DCMIType
format	extent, medium	
identifier	bibliographicCitation	URI
source		URI
language		RFC 4646, ISO639-2, ISO639-3, RFC1766, RFC3066
relation	isVersionOf, isReplacedBy, isRequiredBy, isPartOf, isReferecendBy, isFormatOf, conformsTo, hasVersion, replaces, requires, hasPart, references, hasFormat	
coverage	spatial, temporal	Box, Point, Period, ISO3166, TGN, W3CDTF
rights	AccessRights, license	

Cuadro 2.2: Elementos *Qualified Dublin Core*. [16]

Un encabezado de materia puede estar de una o más palabras. Un encabezado de una palabra representa un concepto mientras que un encabezado múltiple puede representar uno o más conceptos.

Los encabezados de materia se describen en un orden prestablecido (indización pre-coordinada) de acuerdo a 4 subdivisiones [7]:

- Temáticas. Limitan el aspecto temático del encabezado principal a tema en especial.

- Forma. Representa la forma bibliografica, literaria o artística en que el tema de un material es organizado o presentado.

- Geográfica. Representa el origen o la localización del encabezado principal.

- Cronólogica. Representa el período de tiempo del encabezado principal ó la fecha de publicación del recurso de información.

En la práctica, la estructura de un encabezado debe quedar como sigue:

[encabezado principal] – [subdivisión temática] – [subdivisión geografica]

–[subdivisión cronólogica] – [subdivisión de forma]

A continuación un ejemplo:

Geología – Enseñanza – Argentina – 1990-1999 – Informes

Por otro lado, los encabezados tienen entre sí los siguientes tipos de relaciones: equivalencia, jerárquica y asociativa [2].

2.2.3. Estándares para contenidos de datos

Son estándares que norman la sintaxis para escribir los valores que se tienen que llenar cuando se registra un contenido de información. Entre los más importantes tenemos:

a) AACR2 (Anglo-American Cataloguing Rules, Second Edition).

Es un conjunto de reglas de uso bibliográfico que norman la forma como se deben adoptar los encabezamientos. Además permiten determinar los puntos de acceso en un catálogo y brindar pautas para describir distintos tipos de documentos.

2.2.4. Estándares para intercambio de datos

Son estándares legibles por máquinas (machine-readable) para facilitar el intercambio de datos. Entre los más importantes tenemos:

a) MODS

Es un esquema de metadatos desarrollado en el 2002 por la Red de Desarrollo de la Biblioteca del Congreso de los Estados Unidos para diversos propósitos. Es un esquema para portar formatos MARC 21 a XML pero también puede ser usado para implementar la creación de registros bibliográficos. Entre sus usos tenemos: como un formato SRU (Search Retrieval via URL) , como una extensión al estándar METS (Metadata Encoding and Transmission Standard) y para representar metadata para harvesting.

2.3. La indización de recursos de información

2.3.1. Concepto

La indización se refiere al proceso de describir y representar del contenido de un documento usando un número limitado de conceptos extraídos del texto de los documentos (palabras claves) o de vocabularios controlados (descriptores, términos o encabezamientos de materia).

Hay dos formas de indizar, a través de lenguaje natural y a través de vocabulario controlado. La indización mediante lenguaje natural usa técnicas como extracción de palabras claves, nota de contenido y redacción de resúmenes. La indización mediante vocabulario controlado usa técnicas como indización pre-coordinada, indización post-coordinada y clasificación. Entre 1957 y 1962, los estudios Cranfeld mostraron que el uso de la lengua natural ó el uso de vocabularios controlados mostraban la misma eficacia para recuperación de la información. Esta conclusión fue corroborada años después con la proliferación de los catálogos en línea al comprobarse que la lengua natural y los vocabularios controlados no son excluyentes sino complementarios.

Los estudios Cranfeld además identificaron 2 variables para medir el desempeño de los sistemas de recuperación: tasa de precisión y tasa de acierto. La tasa de precisión es la proporción de ítems recuperados que son relevantes para el usuario y la tasa de acierto es la proporción de ítems relevantes que han sido recuperados por el usuario en comparación con el número total disponible. La tabla 2.3 muestra el comportamiento de estas variables con respecto a los tipos de indización.

2.3.2. El proceso de indización

El proceso de indización consiste en tres pasos: el análisis conceptual, la representación o formalización y el almacenamiento en el registro bibliográfico [9].

Variable	Lengua Natural	Vocabularios Controlados
Tasa de precisión	Aumenta por el nivel de especificidad y actualización de los términos	Aumenta por el uso de términos compuestos y el control de homónimos y términos polisémicos
Tasa de acierto	Aumenta cuando se incluyen palabras claves, notas de contenido, resúmenes o texto completo, por la exhaustividad	Aumenta por el control de sinónimos y las relaciones jerárquicas y asociativas.

Cuadro 2.3: Comportamientos de variables de tasa de precisión y acierto. [9]

a) Análisis Conceptual. Consiste en determinar los conceptos o materias que trata el recurso de información que se está procesando. Como resultado, el bibliotecario debe seleccionar aquellos que mejor describan el ítem. Según Taylor hay 4 formas de analizar el contenido de un recurso de información.

Según el propósito. Se identifica el propósito del autor.
Primero y segundo plano. Se identifica una figura central y un contexto de segundo plano.
Modo objetivo. Se toma en cuenta el número de veces que el autor menciona un concepto.
Selección y rechazo. Se determina lo que el ítem dice o no dice (rechazo).

b) Representación o formalización. Después que se han identificado las materias por las que será recuperado el recurso, es necesario estandarizarlo de acuerdo a algún lenguaje artificial como una lista de encabezados de materia por ejemplo en el caso de tratarse de una indización pre-coordinada.

c) Registro en el catálogo. Se refiere al almacenamiento de los datos que conformarán cada registro bibliográfico de acuerdo a ciertas normas.

2.3.3. Herramientas para la indización

a) Listas de encabezamiento de materia

Las listas de encabezamiento de materia son un lenguaje pre-coordinado, de estructura asociativa o combinatoria, que consiste en listas alfabéticas de palabras o

expresiones del lenguaje natural capaces de representar los temas de los que se trata un documento [10].

Los principios básicos sobre los que se asientan las listas de encabezamiento de materia son:

- Especifidad. La elección del término más específico por encima de los más generales.
- Síntesis. La asignación de un encabezamiento de materia es un proceso mental de condensación para la reducción del contenido de un documento al menor número posible de asuntos o materias, y la búsqueda de la expresión con la mayor simplicidad.
- Linguístico. Se emplea el lenguaje usual y en el orden natural del idioma, y prevalece el sustantivo frente al adjetivo, como elemento inicial.
- Economía. La asignación del mínimo número de encabezamientos de materia, por lo general, de uno a tres en una biblioteca pública.

b) Tesauros

Un tesauro esta constituído por terminología normalizada y controlada con el fin de convertir el lenguaje natural empleado en los documentos y en las preguntas a dicha terminología. Los términos incluidos en los tesauros se relacionan entre sí por medio de una serie de símbolos (sintaxis) que establecen relaciones de equivalencia, jerárquicas y asociativas (semántica). La cobertura de los tesauros es, por lo general, de carácter sectorial [4].

- Las relaciones de equivalencia: este tipo de relación de manifiesta con los símbolos UP y USE. UP significa “usado por” . El término que sigue a este símbolo es un sinónimo o cuasinónimo de un descriptor o término preferente. Un término no preferente, también denominado no-descriptor, no se asigna a los documentos, pero proporciona un punto de acceso a partir del cual el usuario es dirigido mediante una instrucción (USE) al término preferente (descriptor). En cambio, el término que sigue al símbolo USE (utilícese) es el preferente entre varios términos sinónimos o cuasinónimos.
- Relación de jerarquía: esta relación se indica con los símbolos TE (término específico) ó NT (narrower term), y TG (término genérico) ó BT (broader term). Según la norma UNE 50-106-90, estas relaciones entre los términos marcan la diferencia entre un tesauro sistemático y una lista no estructurada de términos como un

glosario o un diccionario. Un término superordenado representa un todo o una clase, y los términos subordinados corresponden a sus miembros ó sus partes. El símbolo TG precede al término superordenado, mientras que el símbolo TE precede al término subordinado.

- Relación asociativa: con el símbolo TR (término relacionado) ó RT (related term) se asocian términos que no son equivalentes y no manifiestan ninguna relación jerárquica pero sí "mentalmente hasta el punto de que la conexión entre ellos debe hacerse explícita en un tesauro".

Los tesauros encierran una estructura jerárquica y combinatoria, y el proceso de búsqueda en estos instrumentos, al contrario que en las clasificaciones y en las listas de encabezamiento de materia, se realiza de modo combinatorio y no secuencial. Los costes de elaboración, en conjunto, son mayores en los tesauros que en las clasificaciones y en las listas de encabezamientos.

2.3.4. La indización automática

a) Concepto

Según [8] se define el proceso de indización automática como el uso de computadoras para extraer o asignar términos de indización sin intervención humana, una vez establecidos programas o normas relativas al procedimiento. Entre las ventajas de usar indización automática tenemos:

- Mayor objetividad, puesto que se aplican siempre los mismos parámetros, con lo que se evita la inconsistencia producida por un mismo indizador o la provocada por diferentes profesionales en el análisis del mismo documento.
- La disminución de los errores en la indización repercute positivamente en las bases de datos en el momento de la recuperación de la información.
- Permite una recuperación de los documentos más rica, si bien es cierto que la indización intelectual parece ser más precisa.

b) Herramientas

1) Listas de palabras vacías (stop words)

Se refiere a las palabras sin significado como artículos, pronombres, preposiciones, etc. que son filtradas antes ó después del procesamiento de datos en lenguaje natural. La frase "stop words" y el uso del concepto en su diseño fue idea de Hans Peter Luhn. No hay una lista global de palabras vacías que todas las herramientas

de procesamiento de lenguajes naturales incorporen. Algunas herramientas evitan usarlo específicamente para soportar búsquedas por frase. El uso de un algoritmo de stemming puede reducir parte de la base lógica o dependencia de una lista de palabras vacías a filtrar.

2) Ponderación de términos

Ley de Zipf

Es una ley empírica formulada usando matemáticas estadísticas. Se refiere a que muchos tipos de datos estudiados en las ciencias sociales y físicas pueden ser aproximadas según la distribución de Zipf. En 1949, G.K. Zipf llegó a la conclusión de que en la comunicación hablada o escrita se produce el llamado principio del mínimo esfuerzo. Este principio está relacionado con el recurso de los hablantes y los escritores en una lengua a la repetición de ciertas palabras en lugar de utilizar otras diferentes. Este investigador estableció la siguiente fórmula tras el estudio del libro Ulysses de Joyce.

$$Frecuencia\ x\ clasificación = constante$$

que representa el valor constante que tiene la relación entre la frecuencia de aparición de las palabras y el rango o puesto que éstas ocupan en el orden frecuencial.De esta forma, la palabra más frecuente aparece el doble que la segunda más frecuente, la segunda más frecuente aparece tres veces que la tercera más frecuente y así sucesivamente.

La ley de Zipf (ver figura 2.2) se genera al plotear los datos en un eje de coordenadas donde el eje X es log(valor) y el eje Y es log(frecuencia) y:

N es el número de elementos

k es su rango

s es el valor del exponente que caracteriza la distribución

de acuerdo a la siguiente fórmula:

$$f(k; s, N) = \frac{\frac{1}{k^s}}{\sum_{n=1}^{N} \frac{1}{n^s}}$$

Frecuencia del término

Luhn en 1957 fue el primero en sugerir que la frecuencia de aparición de las palabras en un documento o en una colección tenía que ver con la utilidad de éstas para la indización. Las palabras de frecuencia muy alta (aquellas que se manifestaban en casi todos los documentos) no aportaban carga informativa debido a su carácter

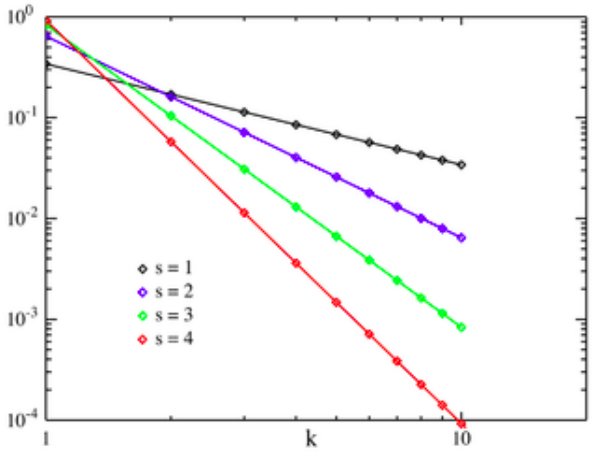

Figura 2.2: La ley de Zipf. Fuente: Wikipedia

general. Por tanto, si se empleaban en la recuperación de la información provocaban una escasa precisión. Por el contrario, los vocablos de la frecuencia muy baja eran muy específicos y causaban una baja exhaustividad en la recuperación. Para Luhn, los mejores términos eran los que presentaban una frecuencia media, es decir, los que no aparecían ni en pocos ni en un gran número de documentos. En 1958 expresó las siguientes consideraciones:

- Dada una colección de n documentos se calcula la frecuencia de aparición de las palabras de cada documento.
- Se determina para cada palabra su frecuencia en toda la colección TOFERQk por la suma de las frecuencias de cada documento.
- Una vez ordenadas las palabras en orden decreciente según su frecuencia en la colección, se eliminan todas las palabras que sobrepasan un umbral de frecuencia determinado. En esta fase se eliminan palabras como: el, de, y, para, a y en, entre otras. En esta fase se prescinde de aquellas poca frecuentes en la colección por medio de un umbral previamente establecido.
- Las palabras restantes, con una frecuencia media, se asignan como términos de indización para los documentos analizados.

Función de frecuencia inversa

Propuesta en 1972 por Sparck Jones, mide la escasez de aparición de un término en una colección. En cierto modo, se toma la idea de que la frecuencia de aparición de una palabra está en relación a su capacidad informativa. Estas ideas están recogidas en la siguiente fórmula.

$$IDF_i = \log_2(\frac{N}{n_i}) + 1 = \log_2(N) - \log_2(n_i) + 1$$

Donde,

N es el número de documentos de la colección y,

n_i es el número de documentos que contienen el término i en la colección.

Valor de discriminación del término

En 1975, Yang y Yu presentaron un nuevo método para conferir el peso ó el valor más alto a aquellos términos que causaban la máxima separación posible entre los documentos de una colección. El valor de discriminación de un término lo definieron como la medida de los cambios manifestados en la separación espacial cuando un término cualquiera es atribuido a una colección como término de indización. En base a estos principios, los términos de indización participan de unas características aproximadas:

- Si se consideran los términos con un valor positivo de discriminación como términos de indización, propicia que decrezca la densidad espacial de los documentos.
- Términos con valores de discriminación indiferentes. Si se suprimen o suman los términos con un valor de discriminación cercano a cero, no cambia la similitud entre los documentos.
- Si se utilizan los términos con valor de discriminación pobre, proporcionan mayores semejanzas entre los documentos, lo que produce un aumento de la densidad espacial de los documentos.

3) KEA (Keyphrase Extraction Algorithm)

KEA es un algoritmo para extraer frases de documentos de texto. Puede ser usado para indexación libre ó para indexación con vocabulario controlado. KEA está implementado en Java y es una plataforma independiente. Es un software libre basado en la licencia GNU.

La tarea de asignar frases claves a un documento es llamado “keyphrase indexing”. Por ejemplo, los papers académicos son frecuentemente acompañados por un conjunto de palabras claves libres escogidas por el autor. En cambio, los indexadores profesionales de bibliotecas seleccionan frases claves de un vocabulario controlado (también llamado Encabezados de Materia) de acuerdo a reglas de catalogación. En el internet, bibliotecas digitales ó cualquier repositorio de datos (flickr, del.icio.us, artículos de blog) también usan frases claves para organizar y proporcionar acceso temático a los datos. A continuación se describe el proceso.

Documentos

KEA identifica un directorio y procesa todos los documentos con extensión *.txt que

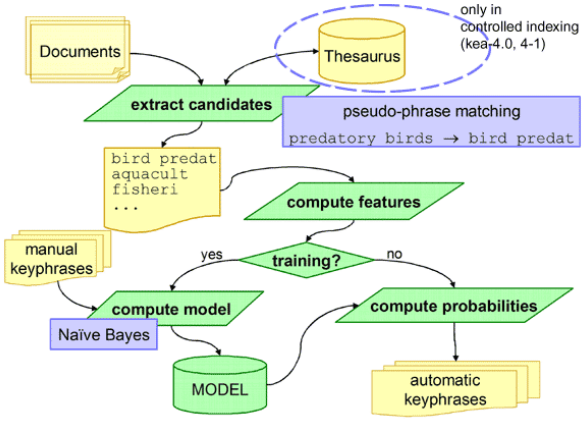

Figura 2.3: Proceso de extracción de frases claves usando el algoritmo KEA. Fuente: Sitio web de KEA

están dentro de este directorio. El lenguaje por defecto y la codificación es el inglés pero puede ser cambiado por el correspondiente archivo “stopword” y un “stemmer”.

Tesauro

Si un vocabulario es proporcionado, KEA coteja las frase del documento contra este archivo. Para procesar los archivos SKOS almacenados como RDF files, KEA usa el API Jena. Para indexación libre, usa la opción “-v none”.

Extracción de candidatos

Aquí KEA extrae n-grams de longitud predefinida (por ejemplo, 1 a 3 palabras) que no comienzan o acaban con un “stopword”. En indexación con vocabulario controlado, solo colecta los n-grams que tienen un equivalente con los términos del tesauro. Si el tesauro define relaciones entre términos no permitidos (“no-descriptores”) y términos permitidos (“descriptores”), reemplazará cada descriptor por su equivalente no-descriptor. En el esquema superior, el término “pseudo-phrase matching” significa remover “stopwords” de las frases claves, para después aplicarle un proceso de stemming y posteriromente ordenar las palabras restantes.

Features

Por cada frase candidata KEA calcula 4 valores importantes:

- TFxIDF es una medida que describe la especificidad de un término para el documento que estamos procesando comparado con otros documentos en el mismo dominio.
- First ocurrence es calculado como el porcentaje del documento precedente a la primera ocurrencia del término en el documento. Los términos que tienden a aparecer al comienzo ó al final del documento son más probables de ser frases claves.

- Longitud de una frase es el número de palabras que lo componen. Frases claves de 2 palabras son preferidos generalmente por indexadores humanos.
- Grado nodal de una frase candidato es el número de frases en el conjunto candidato que están semánticamente relacionados con la frase. Esto es calculado con la ayuda del tesauro. Frases con alto grado nodal son más probable de ser frases claves.

Construyendo el modelo

Antes de ser capaz de extraer frases claves de nuevos documentos, KEA necesita primero crear un modelo que aprenda como extraer estratégicamente tomando como referencia la indexación manual de otros documentos. Esto quiere decir, que por cada documento en el directorio de entrada debe haber un archivo con extensión ".key" con el mismo nombre que el documento correspondiente. Este archivo deberá contener las frases claves que fueron asignadas manualmente, una por línea. Dada la lista de frases candidatas (ver punto 3), KEA marca las que fueron manualmente asignadas como un ejemplo positivo y todo el resto como ejemplos negativos. Al analizar los valores claves (ver punto 4) para las frases candidatas positivas ó negativas, un modelo es calculado, que refleje la distribución de los valores claves por cada frase.

Extracción de frases

Cuando extrae frases claves de nuevos documentos, KEA toma el modelo (ver punto 5) y los valores claves por cada frase candidata y calcula la probabilidad de ser una frase clave. Frases con la más alta probabilidad son seleccionadas dentro del conjunto final de palabras claves. El usuario puede especificar el número de frases claves que necesitan ser seleccionadas.

CAPÍTULO 3

LA WEB SEMÁNTICA

3.1. Conceptos Generales

La web semántica es una web de datos. La visión de la web semántica es extender los principios de la web de documentos a datos. Los datos deberían ser accedidos usando la arquitectura general web, por ejemplo URIs; los datos deberían ser relacionados uno a otros como los documentos lo están ahora. Esto también significa la creación de un framework común que permita a los datos compartidos y reusados por aplicaciones, empresas y la sociedad en general ser procesados automáticamente por herramientas así como manualmente de modo que revelen nuevas posibles relaciones entre piezas de datos.

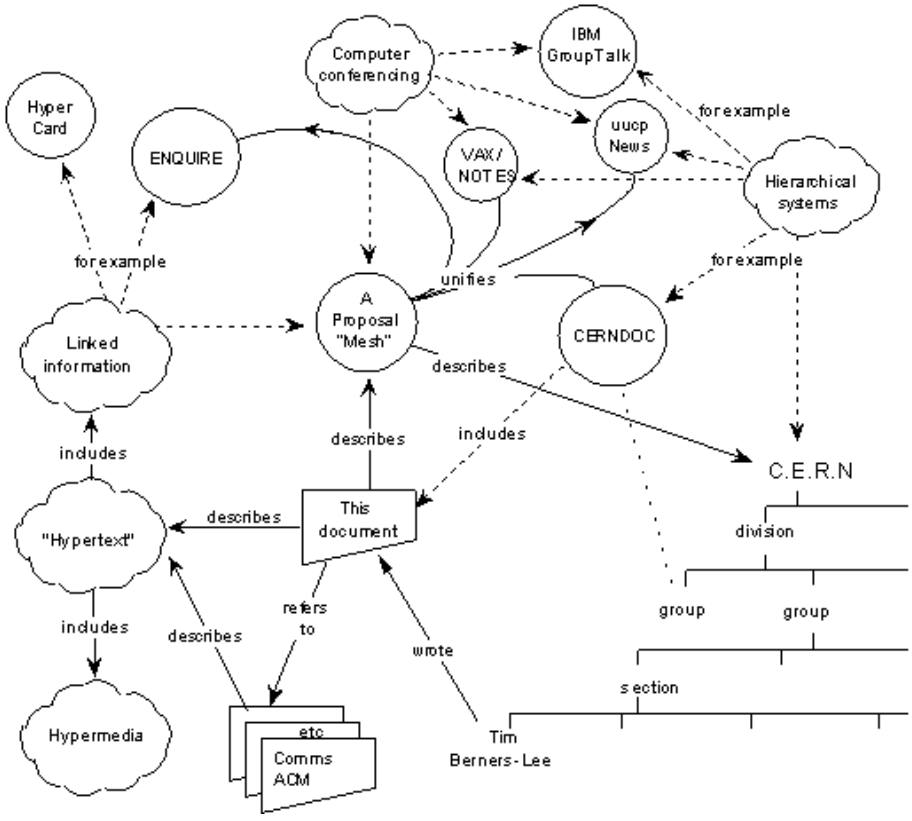

Figura 3.1: Visión de la web según su creador Tim Berners Lee en 1999.
Fuente: Sitio web W3C

Para lograr estos objetivos, es necesario definir y describir las relaciones entre los datos en la web. Esto no es diferente al uso de enlaces que conectan la página actual con otra página: los enlaces definen la relación entre la página actual y su objetivo. En la web semántica, tales relaciones pueden ser establecidas entre dos

recursos si asignamos un identificador a cada uno de ellos y a la relación que tienen ambos entre sí.

Según [15] la web semántica puede ser usada en una variedad de áreas:

En integración de datos mediante el cual los datos ubicados en diferentes lugares pueden ser integrados en uno.

En clasificación y descubrimientos de recursos porque mejora las capacidades de los motores de búsqueda en dominios específicos.

En catalogación para describir el contenido y sus relaciones disponibles en un sitio web o en una biblioteca digital.

En software inteligente para facilitar el intercambio de contenidos.

En clasificación de contenido para describir colecciones de páginas que representan un único documento lógico.

En derechos de propiedad intelectual

3.1.1. Arquitectura

Desde su propuesta inicial, Tim Berners Lee, el inventor de la web, ha propuesto tres versiones de la arquitectura de la web semántica. La primera versión fue presentada en el 2000 en la conferencia XML2000, la segunda en la conferencia "Software & Information Industry Association" (SIIA) del 2003, la tercera en la conferencia WWW2005 y la cuarta en la conferencia AAAI2006. La figura 3.2 muestra el estado de la última versión.

A continuación describimos en qué consiste cada parte del esquema.

a) Uniform Resource Identifier (URI)

Es una cadena de caracteres que identifica un recurso físico o abstracto en internet por su localización, por su nombre ó por ambos. La especificación de su sintaxis y semántica es derivada de conceptos introducidos por la World Wide Web y cuyo uso data de 1990. El URI está caracterizado por las siguientes definiciones:

Uniforme

La uniformidad proporciona algunos beneficios: permite usar diferentes tipos de identificadores de recursos en el mismo contexto, incluso cuando los mecanismos usados para acceder a estos recursos son diferentes. Permite identificación semántica uniforme de convicciones sintácticas comunes a través de diferentes identificadores de recursos. Permite la presentación de nuevos tipos de identificadores de recursos sin interferir en la forma como trabajan los identificadores existentes y permite que los identificadores sean reusados en muchos diferentes contextos, permitiendo que las nuevas aplicaciones aprovechen las existentes.

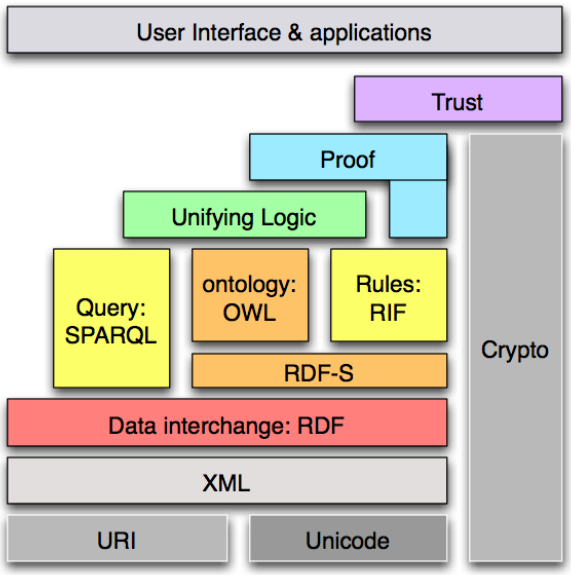

Figura 3.2: Arquitectura de la web según Berners Lee.
Fuente: Sitio web W3C

Recurso

Puede ser cualquier cosa que tenga identidad. Como ejemplos podríamos documentos electrónicos, un servicio de reportes de clima ó una colección de otros recursos. No todos los recursos son recuperables por red, seres humanos, empresas, y libros de biblioteca. El recurso es un mapeo conceptual a una entidad ó a un conjunto de entidades, no necesariamente la entidad que corresponde a ese mapeo en diferentes intervalos de tiempo. Así, un recurso puede permanecer incluso cuando su contenido cambia a lo largo del tiempo, siempre que el mapeo conceptual no se cambie durante el proceso.

Identificador

Un identificador es un objeto que puede actuar como referencia a algo que tiene identidad. En el caso del URI, el objeto es una secuencia de caracteres con una sintaxis restringida.

Un URL puede ser además clasificado como un localizador, un nombre ó ambos. El término *Uniform Resource Locator* (URL) se refiere a un subconjunto de URIs que identifican recursos vía una representación de sus mecanismos de acceso primario más que por su nombre ó por otro atributo de ese recurso. Por otro lado, el término *Uniform Resource Name* (URN) se refiere al subconjunto de URIs que son requeridos para permanecer únicos y persistentes incluso cuando el recurso cesa de existir ó queda no disponible.

b) Unicode

Es un estándar que proporciona un número único por cada carácter sin importar el idioma, sin importar el programa ó la plataforma. Ha sido adoptado por importantes líderes de la industria como Apple, HP, IBM, JustSystem, Microsoft, Oracle, SAP, Sun, Sybase, Unisys, etc. Es un requisito para otras tecnologías como XML, Java, ECMAScript (JavaScript), LDAP, CORBA 3.0, WML, etc. Además es la manera oficial de aplicar la norma ISO/IEC 10646.

La última versión cuenta con 109 000 caracteres y cubre 93 scripts, un conjunto de tablas de código para referencia visual, una metodología de codificación y un conjunto de estándares, una enumeración de propiedades de caracteres para mayúsculas y minúsculas, reglas para normalización, descomposición, cotejamiento, renderizado y orden de display bidireccional para lenguajes como el árabe y hebreo.

Es compatible con muchos sistemas operativos, con todos los exploradores actuales y con muchos otros productos. La aparición de la norma Unicode y la disponibilidad de herramientas que la respaldan, se encuentran entre las más recientes e importantes tendencias en tecnología de software.

La incorporación de Unicode en sitios Web y en aplicaciones de cliente-servidor o de múltiples niveles permite disminuir ostensiblemente los costos del uso de juegos de caracteres heredados. Unicode permite que un producto de software o sitio Web específico se oriente a múltiples plataformas, idiomas y países sin necesidad de rediseñarlo. Además permite que los datos se trasladen a través de muchos sistemas distintos sin sufrir daños.

Unicode puede ser implementado en diferentes sistemas de codificación. Los más famosos son UTF-8 y UTF-16.

c) XML

XML (*Extensible Markup Language*) describe una clase de objetos de información llamados documentos XML y parcialmente describe el comportamiento de programas de computadora que lo procesan. Es un perfil de aplicación o forma restringida de SGML (*Standard Generalized Markup Language*).

Los documentos XML están hechos de unidades de almacenamiento llamado entidades que contienen datos analizados y sin analizar. Los datos analizados están hechos de caracteres, algunos de los cuales conforman el contenido y otros el markup (tags). El markup codifica una descripción de las capas de almacenamiento de

documentos y su estructura lógica. XML proporciona un mecanismo para imponer límites sobre las capas de almacenamiento y la estructura lógica.

Cada documento XML tiene una estructura física y lógica. Físicamente, el documento está compuesto de unidades llamadas entidades. Una entidad puede referirse a otras entidades que originan su inclusión en el documento. Un documento empieza en una raiz o llamada también entidad de documento. Logicamente, el documento está compuesto de declaraciones, elementos, comentarios, referencias de caracteres e instrucciones de procesamiento. Todos ellos son indicados en el documento con un explícito lenguaje de marcas (markup) . La estructura lógica y física debe anidarse apropiadamente.

XML fue desarrolllado bajo los auspicios de la World Wide Web Consortium (W3C) en 1996. Fue liderado por Jon Bosak de Sun Microsystems.

Fue diseñado con los siguientes objetivos:

- Simple de usar en internet.
- Soportar una amplia variedad de aplicaciones.
- Ser compatible con SGML.
- Fácil de implementar programas que procesen documentos XML.
- El número de características opcionales en XML debe ser mínima, idealmente cero.
- Los documentos XML deben ser legibles por humanos y razonablemente claros.
- El diseño de XML debe ser preparado rápidamente.
- El diseño de XML debe ser formal y conciso.
- Los documentos XML deben ser fácil de crear.

3.2. Infraestructura de Descripción de Recursos

Según [14], la Infraestructura de Descripción de Recursos (RDF, *Resource Description Framework*) es un lenguaje de modelamiento semántico que surge como respuesta a la necesidad de modelar objetos reales y sus relaciones en el mundo virtual. Entre las ventajas que ofrece están:

- Proporciona información sobre recursos web y los sistemas que les dan soporte (ranking de contenidos, opciones de privacidad, etc).
- Permite el desarrollo de aplicaciones abiertas (calendario de actividades, descripción de procesos organizacionales, anotación de recursos web, etc).

- Permite que todo el contenido publicado en la web sea procesado a gran escala por las computadoras independientemente del lugar donde fue creado.
- La creación de nueva información a partir de la combinación de datos provenientes de diferentes aplicaciones.
- Proporciona un lenguaje común para que los agentes de software escaneen y procesen la web sin problemas.

3.2.1. Características

- RDF tiene un modelo simple de datos que simplifica el trabajo de manipulación y procesamiento de las aplicaciones de software. Este modelo de datos es independiente de cualquier sintaxis específica.
- RDF tiene una semántica formal muy consistente que facilita la construcción de expresiones a partir de las cuales las computadores pueden inferir lógicas más complejas con facilidad.
- RDF usa las reglas del XML como sintaxis para representar el modelado de los objetos y URIs para identificarlos únivocamente.
- RDF es compatible con otros esquemas basados en XML facilitando así el intercambio de información.
- RDF facilita la descripción de cualquier recurso por cualquier usuario ya que es un estándar abierto.

3.2.2. Conceptos Básicos

a) Modelo de Datos basado en gráficos

La unidad básica se llama tripleta y está compuesta de tres elementos: un sujeto, un objeto y un predicado (también llamado propiedad) que denota la relación entre los dos tal como se muestra en la figura 3.3:

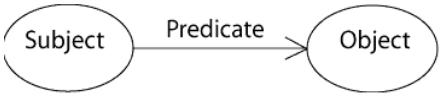

Figura 3.3: Ejemplo de un grafo RDF.
Fuente: Sitio web W3C

La dirección de la flecha es importante y siempre apunta hacia el objeto. Este enfoque simple permite que los datos estructurados o semi-estructurados sean combinados y compartidos entre diferentes aplicaciones. A los sujetos y objetos de un gráfico RDF se les llama nodos.

b) Identificación de nodos basada en URIs

Un nodo puede ser un URI con un identificador opcional de fragmentos, un valor literal ó un valor en blanco. Las propiedades también pueden ser URIs. En cualquier caso los URIs no pueden relativos.

Un URI ó un valor literal usado como un nodo identifica lo que el nodo representa. Por otro lado un URI usado como predicado representa una relación entre los objetos representados por los nodos que conecta.

Un nodo en blanco es un nodo que no es un URI ni un valor literal. En la sintaxis abstracta del RDF, un nodo en blanco es sólo un nodo único que puede ser usado en una ó más sentencias RDF pero no tiene un nombre intrínseco.

c) Tipos de Datos

Los tipos de datos son usados para la representación de valores tales como enteros, números de punto flotante y fechas. Un tipo de datos consiste de un espacio léxico, un espacio valor y un mapeo entre ellos. Por ejemplo, para el tipo de dato xsd:boolean de XML Schema el mapeo sería como se muestra en la tabla 3.1.

Espacio Valor	{T, F}
Espacio Léxico	{"0", "1", "true", "false"}
Mapeo entre espacio léxico y espacio valor	{<"true", T>,<"1",T>,<"0",F>,<"false",F>}

Cuadro 3.1: Valores para el tipo de dato xsd:boolean de XML Schema

d) Valores Literales

Los valores literales son usados para identicar valores tales como números y fechas por medio de una representación léxica. Cualquier cosa representada por un literal

podría también ser representada por un URI, pero es más conveniente ó intuitivo usar valores literales. Un valor literal puede ser el objeto de una sentencia RDF pero no el sujeto ni el predicado.

e) Espacios de nombres (namespaces)

Existen ciertos URIs que han sido reservados ya que definen especificaciones RDF. Por ejemplo, el URI http://www.w3.org/1999/02/22-rdf-syntax-ns# está reservado porque corresponde al espacio de nombres XML llamado XML-NS asociado con los términos del vocabulario RDF. Convencionalmente está asociado con el prefijo "rdf:". Algunos términos definidos por las especificaciones RDF denotan conceptos específicos y otros sólo tienen propósito sintáctico.

3.3. Esquema RDF (RDFS)

Según [13] RDF Schema es una extensión semántica de RDF. Proporciona mecanismos para describir grupos de recursos relacionados y la relación entre estos recursos. Estos recursos son usados para determinar características de otros recursos tales como los dominios y los rangos de las propiedades.

El sistema de clases y propiedades de RDF Schema es similar a los sistemas de programación orientada a objetos como Java. Difiere de otros sistemas porque en lugar de definir una clase en términos de las propiedades que sus instancias pueden tener, describe propiedades en términos de las clases de recursos sobre las que está trabajando.

Esta especificación no pretende enumerar todas las posibles formas para la descripción de vocabularios que son útiles para representar el significado de clases RDF y sus propiedades. En su lugar, la estrategia de descripción de los vocabularios RDF parte de reconocer que hay muchas técnicas a través de las cuales el significado de clases y propiedades puede ser descrito. Ontologías como DAMN + OIL, el lenguaje OWL de la W3C, los lenguajes de reglas de inferencia y otros formalismos contribuirán a nuestra capacidad para capturar generalizaciones significativas sobre datos en la Web. Los diseñadores de vocabularios RDF pueden crear y desplegar aplicaciones para la Web Semántica usando las facilidades que el lenguaje de descripción de vocabularios RDF puede brindar.

El lenguaje definido en esta especificación consiste de una colección de recursos RDF que pueden ser usados para describir propiedades de otros recursos RDF (incluyendo propiedades) en vocabularios RDF de aplicaciones específicas. El vocabulario del núcleo es definido como un Espacio de Nombres informalmente llamado 'rdfs' . Ese Espacio de Nombres es identificado por la referencia URI

http://www.w3.org/2000/01/rdf-schema# y está asociado con el prefijo 'rdfs'. Esta específicación también usa el prefijo 'rdf' para referirse al Espacio de Nombres RDF http://www.w3.org/1999/02/22-rdf-syntax-ns#.

Para conveniencia y legibilidad, ésta especificación usa un forma abreviada para representar referencias URI. Un nombre de la forma prefix:suffix debería ser interpretada como una referencia URI que consiste de una referencia URI concatenada con el sufijo.

3.4. Lenguajes de Ontología Web (OWL)

Según [6] la Web Ontology Language (OWL) extiende RDF y RDFS. Su objetivo primario es traer el poder de razonamiento y expresividad de la lógica descriptiva (DL) a la web semántica. Desafortunadamente, no todo el RDF puede ser expresado en DL. Por ejemplo, las clases de clases no son permitidas en DL, y algunas de las expresiones de las tripletas no tienen sentido en DL. Eso es por qué OWL can ser sólo extensión sintáctica de RDF/RDFS. Para superar parcialmente este problema, y también para permitir la creación de capas dentro de OWL, 3 tipos de OWL son definidos.

3.4.1. Los tres sublenguajes de OWL

Según [12] OWL proporciona tres sub lenguajes expresivos diseñado para ser usado por comunidades específicas de implementadores y usuarios:

a) OWL Lite soporta a usuarios que primariamente necesitan una clasificación jerárquica y restricciones simples. Por ejemplo, aunque es comoatible con las restricciones de cardinalidad, sólo permite valores de cardinalidad de 0 ó 1. Debería ser simple para proporcionar una herramienta de soporte a OWL Lite . OWL Lite ofrece una ruta de migración rápida para tesauros y otras taxonomías.

b) OWL DL apoya a los usuarios que quieren la máxima expresividad conservando la completitud del cálculo (se garantiza que todas las conclusiones sean computadas) y decidibilidad (todos los cálculos acabarán en un tiempo finito). OWL DL incluye todas las construcciones del lenguaje OWL, pero puede ser utilizado solamente bajo ciertas restricciones (por ejemplo, mientras que una clase puede ser una subclase de muchas clases, una clase no puede ser instancia de otra clase). OWL DL se llama así debido a sus correspondencia con lógicas de descripción, un campo de investigación que ha estudiado las lógicas que conforman la base formal de OWL.

c) OWl Full está pensado para usuarios que desean máxima expresividad y la libertad sintáctica del RDF sin garantías computacionales. Por ejemplo, en OWL Full una clase puede ser tratada simultáneamente como una colección de individuos y como un individuo por propio derecho. OWL Full permite una ontología para aumentar el significado de vocabularios (RDF ó OWL) predefinidos. Es poco probable que cualquier sotftware de razonamiento sera capaz de soportar el razonammiento completo para característica de OWL Full.

3.5. Datos Vinculados (*Linked Data*)

La web nos ayuda a enlazar documentos. Similarmente nos puede ayudar a enlazar datos. Según la página de la comunidad Linked Data1, el término Linked Data se refiere a un conjunto de prácticas para publicar y conectar datos estructurados en la web. Tecnologías claves que soportan Linked Data son URIs (un identificador de entidades ó conceptos en el mundo), HTTP (un mecanismo simple para recuperar documentos ó descripción de recursos), y RDF (un modelo de datos basado en grafos para describir las cosas que hay en el mundo).

CAPÍTULO 4

DESARROLLO DE APLICACIONES WEB CON DRUPAL

4.1. Arquitectura General

Drupal es ideal para construir sitios web. Es altamente modular, un framework de administración de gestión de contenidos basado en software libre con un énfasis en colaboración. Es extensible, compatible con estándares, con código limpio y muy ligero. La versión básica de Drupal consiste en un core con funcionalidades básicas que se puede extender fácilmente con módulos desarrollados por terceros. Drupal está diseñado para ser personalizada. Esta personalización se realiza al sobrescribir el núcleo o al agregar módulos sin necesidad de sobrescribir el código . El diseño de Drupal también separa exitosamente el sistema de administración de contenidos de la capa de presentación.

Drupal puede ser usado para construir un portal de internet; un website personal, departamental o corporativo; un sitio de comercio electrónico; un directorio de recursos, un periódico online, una galería de imágenes, una intranet, para mencionar algunas cuantas posibilidades. Incluso puede ser usado para cursos a distancia.

Un equipo de seguridad se dedica a solucionar amenazas y generar actualizaciones de seguridad. Una organización llamada Drupal Association brinda el soporte en infraestructura a la web principal drupal.org y se encarga de organizar conferencias Drupal y eventos.

Drupal está diseñado para trabajar en hostings baratos y para ser escalable hasta sitios de distribución masiva. Su arquitectura se puede ver en la figura 4.1.

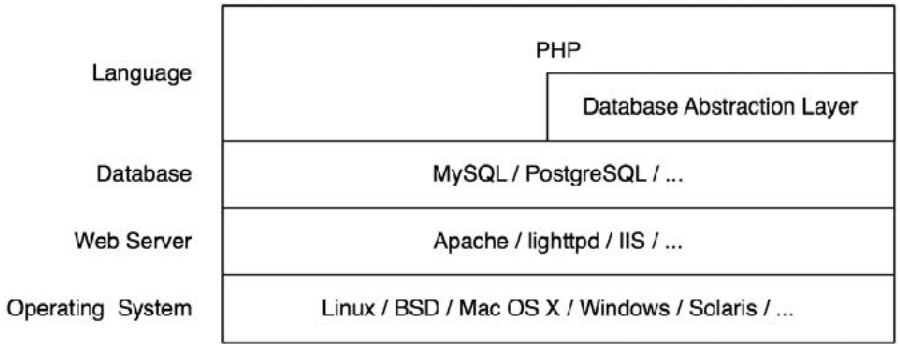

Figura 4.1: Arquitectura de implementación de Drupal.
Fuente: [11]

El sistema operativo se encuentra en el más bajo nivel y es totalmente independiente a Drupal. Drupal corre sucesivamente en cualquier sistema operativo que

soporta PHP.

El servidor más usado con Drupal es Apache pero también pueden usarse otros servidores web como IIS. Debido a su vínculo con Apache, Drupal se basa en el archivo .htaccess para la seguridad de su instalación. Esto también permite que los URLs sean limpios, es decir legibles gracias a la interacción con el modulo mod_rewrite del Apache. Esto es particularmente importante porque cuando se migran contenidos de otros sistemas de gestión de contenidos se necesita que los URLs puedan ser personalizados de modo que mantengan su permanencia (*permalinks*).

Drupal se comunica con la siguiente capa del esquema a través de una capa liviana de abstracción de base de datos. Esta capa controla todas las consultas SQL y hace posible que Drupal interactúe con todo tipo de motores de bases de datos sin tener que modificar la estructura del código. Trabaja de forma nativa con MySQL y Postgre-SQL, aunque está creciendo el soporte para Microsoft y Oracle.

Drupal está escrito en PHP. Desde que PHP es un lenguaje fácil de aprender, hay mucho código escrito por usuarios aficionados. La calidad del código programado por estos usuarios le ha dado mala reputación a PHP. Sin embargo, PHP también puede ser usado para escribir un código sólido. Todo el núcleo de Drupal ha sido escrito de acuerdo a los más altos estándares y revisado estrictamente de acuerdo al estilo de la comunidad de software libre.

El núcleo incluye código que le permite a Drupal recorrer todo un ciclo cuando recibe una petición, una librería de funciones comunes frecuentemente usadas con Drupal, y módulos que proporcionan funcionalidad básica como administración de usuarios, taxonomías y sistemas de plantillas tal como se muestra en la figura 4.2.

4.2. Interfaz de programación de aplicaciones (API)

4.2.1. Funcionamiento interno de Drupal

Drupal se arranca a sí mismo en cada request a través de una serie de fases de arranques. Estas fases están definidas en el archivo boostrap.inc y se describen a continuación:

a. Inicialización de la configuración

Esta fase inicializa toda la configuración de Drupal en un arreglo y establece el url base del sitio web. El archivo de configuración settings.php es analizado vía la función include_once() y cualquier variable ó cadena definida en este archivo sobreescribe alguna que este definida en el sitio web.

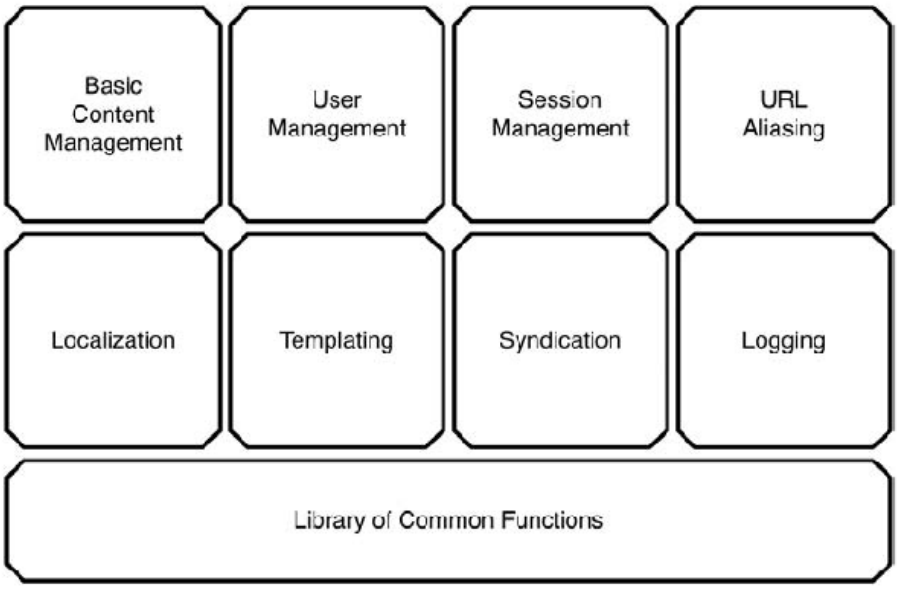

Figura 4.2: Arquitectura modular de Drupal.
Fuente: [11]

b. Inicialización de la caché de páginas

En situaciones donde se requiera un alto nivel de escalabilidad, un sistema de caché puede ser necesario antes de establecer una conexión a base de datos por cada carga de página.

c. Inicializar base de datos

Durante la carga de la base de datos, el tipo de base de datos es detectado y se establece una conexión inicial que será usada para consultas a base de datos.

d. Control de acceso basado en nombre de host ó IP

Drupal permite el control de acceso de hosts en base al control de los nombres de hosts o IPs. En la fase de control de acceso, un chequeo rápido es hecho para ver si la petición viene de un host bloqueado, si se cumple entonces el acceso es denegado.

e. Inicializando control de sesión

Drupal aprovecha el control de sessión de PHP pero sobrescribe algunos de sus parámetros de control con su propia implementación de control de sesión basada en base de datos. Las sesiones son inicializadas y reestablecidas en cada fase de sesión. El objeto global $user que representa al usuario actual es también inicializado aquí, aunque por eficiencia no todas sus propiedades estan disponibles (ellas son agregadas cuando se hace una llamada explícita a la función user_load()).

f. Reconfiguración de la caché

En esta fase, Drupal carga suficiente código para determinar si sirve o no una página desde la caché. Esto incluye la combinación de las configuraciones que viene de la base de datos con las que estaban guardadas en el arreglo cuando empezó la fase de arranque. Si la sesión indica que la petición fue hecha por un usuario anónimo y el sistema de caché está activado, la página es extraída de la caché y el proceso de ejecución se detiene.

g. Determinación del lenguaje

En la fase de determinación del lenguaje, el soporte multilenguaje es inicializado y la página es lanzada en el lenguaje adecuado de acuerdo a la configuración del sitio web y del usuario. Drupal provee diferentes alternativas para determinar el lenguaje, tales como prefijo de ruta y negociación de lenguaje a nivel de dominio.

h. Ruta

En la fase de ruta, el código que controla las rutas y sus alias es cargado. Esta fase activa los urls legibles por ser humanos que serán mapeadasinternamente con las rutas cacheadas de los urls de Drupal.

i. Total

Esta fase completa el proceso de arranque al cargar una librería de funciones comunes, soporte de plantillas y soporte de mapeo de llamadas, gestión de archivos, UNICODE, herramientas de manipulación de imágenes, creación y manipulación de formularios, gestión de correos y paginación de resultados. El controlador de errores personalizados de Drupal es configurado y todos los módulos son cargados. Finalmente, rupal llama al hook init, de forma que todos los módulos tengan la oportunidad de ser notificados antes que el procesamiento de la petición oficial empiece.

Una vez que Drupal ha completado el proceso de arranque, todos los componentes del framework estarán disponibles. En ese momento toma la petición del navegador y se lo entrega a la función de php que lo manejará. El mapeo entre las funciones y los URLs está registrada y sirve para controlar el acceso. Cuando Drupal recibe una petición desde el navegador y determina que existe una función correspondiente entonces le entrega el control luego de verificar el acceso.

4.2.2. Librerías del código de Drupal

Las librerías de Drupal están organizadas de acuerdo a los siguientes componentes:

a. Sistemas de módulos

Esta librería permite que los módulos interactuén con el núcleo de Drupal.

El sistema de módulos está basado en el concepto de hooks (ganchos). Un hook, es una función PHP que es nombrado de la forma foo_bar(), donde “foo” es el nombre del módulo (cuyo archivo que le contiene lleva por nombre foo.module) y “bar” es el nombre del hook. Cada hook tiene un conjunto definido de parámetros y un tipo específico de resultado.

Para extender Drupal, un módulo necesita simplemente implementar un hook. Cuando Drupal desea permitir la intervención de los módulos, determina que módulos implementan un hook y llama a todos los módulos que tienen implementados ese hook.

b. Capa de abstracción de base de datos

Esta librería permite el uso del mismo código base por diferentes servidores de bases de datos. Drupal proporciona una capa de abstraccion de base de datos muy ligera que le permite a los desarrolladores la capacidad de implementar soporte para varias bases de datos sin problemas. El propósito de esta capa es preservar la sintaxis y el poder de SQL tanto como sea posible, dejándole a Drupal el control de las consultas que necesitan escribirse para diferentes servidores y con la seguridad pertinente.

Muchas consultas a la base de datos son realizadas a través de la función db_query() ó db_query_range(). Los autores de módulos también considerar usar la función pager_query() para consultas que devuelvan resultados que necesitan ser presentados en páginas multiples, y la función tablesort_sql() para generar consultas apropiadas para tablas ordenables. Por ejemplo, si uno desea obtener una lista de los 10 nodos más recientes de un autor. En lugar de hacer:

```
<?php
  "SELECT n.nid, n.title, n.created FROM node n WHERE
  n.uid = $uid LIMIT 0, 10";
?>
```

Se debería escribir lo siguiente de acuerdo al API de Drupal

```
<?php
  $result = db_query_range("SELECT n.nid, n.title , n.created
    FROM {node} n WHERE n.uid = %d", $uid, 0, 10);
  while ($node = db_fetch_object($result)) {
    // Perform operations on $node->body, etc. here.
  }
?>
```

Las llaves son usadas alrededor del nombre de la tabla node para proporcionar de manera automática el prefijo vía la función db_prefix_tables(). El uso explícito del id de un usuario es encapsulado para evitar ataques de inyección SQL. Por último, como la sintaxis de la claúsula LIMIT varía entre servidores de base de datos, entonces se debe usar la función db_query_range().

c. El sistema de menúes

Define los menúes de navegación y las rutas de las peticiones basadas en URLs.

El sistema de menús Drupal controla el sistema de navegación desde la perspectiva de usuario y el sistema de devolución de llamadas que Drupal usa para responder a los URLs que provienen del browser. Por esta razón, es fundamental entender el comportamiento de este sistema para construir sistemas complejos.

El sistema de menúes de Drupal sigue una simple jerarquía definida por rutas. La implementaciones de la función hook_menu define items de menú y los asigna a las rutas (que deberían ser únicas). El sistema de menúes agrega estos items y determina la jerarquía de menú para estas rutas. Por ejemplo, si las rutas definidas fueron a, a/b, a/b/c/d, f/g, el sistema de menúes tendría la siguiente estructura:

```
-a
  -a/b
       -a/b/c/d
       -a/b/h
-e
-f/g
```

Note que el número de elementos de cada ruta no necesariamente determina la profundidad del menú ítem en el árbol. Cuando responde a una petición de una página, el sistema de menúes busca si la ruta requerida por el navegador esta registrada como ítem de menú con una devolución de llamada. Si no lo encuentra, revisará todo el árbol hasta encontrar uno compatible. Si el path a/b/i es requerido en el árbol arriba y no lo encuentra, entonces usará la ruta a/b.

La función encontrada de devolución de llamadas es llamada con argumentos especificados en el atributo “page arguments” de su menú ítem. El atributo debe estar en un arreglo. Después de estos argumentos, el resto de argumentos es agregado como argumentos adicionales. De esta forma, la devolución de llamada para a/b arriba podría responder a una petición para a/b/i diferente mas que a una petición para a/b/j.

El acceso a las funciones de devolución de llamada es también protegida por el sistema de menúes. El atributo “access callback” con un atributo opcional .ªccess arguments"de cada menú es llamado antes que proceda la llamada a la página. Si devuelve TRUE, entonces el acceso es concedido; si devuelve FALSE, entonces el acceso es denegado. Los menúes de items pueden omitir este atributo para usar el valor proporcionado por un ítem antecesor.

En la interfase de Drupal por defecto, se pueden ver algunos links renderizados como tabs. Estos son conocidos en el sistema de menúes como “local tasks”, y ello son renderizados como tabs por defecto, aunque otras presentaciones son posibles. La función “local tasks” funciona como otros menúes en otros aspectos. Es convención que los nombres de estos tabs deberían ser verbos cortos en la medida de lo posible. Además, un “local task” por defecto debe ser proporcionado por cada grupo de tabs. Cuando se visita un menú item raíz de un “local tasks”, el tab por defecto ("default local task") será renderizado tal como fue seleccionado. Esto facilita la experiencia de usuario al proporcionarsele un tab por defecto. Todo lo descrito se almacena en la tabla menu_router. La tabla menu_links contiene todos los enlaces visibles de menúes.

Por defecto, los menúes son creados a través de la función hook_menu pero también pueden ser creados por la función menu_link_save().

d. Generación de formularios

Define las funciones para permitir el procesamiento y visualización de los formularios HTML. Drupal usa estas funciones para lograr consistencia en la presentación y presentación de formularios al simplificar código y reducir la cantidad de HTML que debe ser explicitamente generado por los módulos.

La función drupal_get_form() controla, recupera, procesa y renderiza los formularios HTML. Por ejemplo:

```
<?php
// Display the user registration form.
$output = drupal_get_form("user_register");
```

?>

Los formularios pueden ser construídos y enviados programadamente sin usar datos de entrada usando la función drupal_execute().

e. Sistema de archivos

Contiene las siguientes funciones para la administración de archivos

file_check_directory Revisa si un directorio existe y si es escribible.

file_check_location Revisa si un archivo está realmente localizado dentro de la variable directory.

file_check_path Revisa la ruta para ver si es un archivo ó un folder

file_copy Copia los archivos a una nueva localización

file_create_filename Crea una ruta completa para un directorio ó un archivo. Si el archivo con el nombre especificado ya existe, una ruta alternativa será usada.

file_create_url Crea la ruta de descarga a un archivo

file_delete Borra un archivo

file_destination Determina la ruta destino para un archivo dependiendo de cómo se ha implementado el control de los archivos a reemplazar

file_directory_temp Determina el directorio de archivos por defecto.

file_download Llama a los módulos que implementan la función hook_file_download para encontrar si un archivo es accesible o que cabeceras deberian ser transferidos con él. Si el módulo devuelve -1 entonces se usará la función drupal_access_denied.

file_get_mimetype Determina el tipo MIME de un documento

file_move Mueve un archivo a una nueva localización.

f. Sistema de búsqueda

La interfase de búsqueda de Drupal administra un mecanismo de búsqueda global.

Los módulos pueden conectarse hacia este sistema para proporcionar búsquedas de diferentes tipos de datos. Todo este sistema es gestionado por las funciones almacenadas en el archivo search.module, de modo que este módulo debe estar activado para que los módulos que implementen mecanismos de búsqueda funcionen.

Existen 3 formas de interactuar con el sistema de búsqueda:

- Para búsquedas específicas en nodos, se puede implementar la función hook_nodeapi('update index') y la función hook_nodeapi('search result'). Sin embargo, la vistas de texto completo de un nodo que hayan usado la función hook_view() para su visualización siempre son indexadas por defecto.
- Implementar la función hook_search(). Esto creará un tab para tu módulo en la página de búsqueda con un formulario de búsqueda simple.
- Implementar la función hook_update_index(). Esto permitirá a tu módulo usar el mecanismo de indexación para HTML de Drupal eficientemente.

g. Sistema de accesos de nodos

El sistema de acceso a nodos determina que operaciones los usuarios pueden hacer con los nodos. La función node_access() es responsable de determinar los permisos para un nodo. Para eso primero revisa si el usuario tiene el permiso "administer nodes". Tales usuarios tienen acceso irresctricto a todos los nodos. Después la función hook_access del módulo node es llamado, y devolverá TRUE ó FALSE de acuerdo a los permisos. Esto permite, por ejemplo, que el módulo Blog siempre le pueda dar acceso al autor de un post ó que al módulo book siempre se le niegue permiso para editar PHP.

Si el módulo node no interviene (devuelve NULL) entonces la tabla node_access es usada para determinar el acceso. Todos los modulos de acceso a nodos usan la función hook_node_grants() para ensamblar una lista de IDs con permisos para el usuario. Esta lista es comparada contra la tabla. Si alguna fila contiene el ID del nodo en cuestión (or 0 que significa para todos los nodos), uno de los IDs de permisos es devuelto, y un valor de TRUE para la operación en cuestión.

h. Sistemas de plantillas

Las funciones y plantillas para la interfase de usuario son implementados por temas (themes).

La capa de presentación de Drupal es un sistema conocido como capas de temas (themes). Cada tema puede tomar control sobre la forma como Drupal presenta finalmente la información. Por otro lado, tiene un dominio completo sobre el css. Dentro de Drupal, la capa de temas es utilizada al usar la función theme() cuando se pasa el nombre del componente (el hook theme) y un arreglo de variables. Por ejemplo,

```
<?php
theme('table', array('header' => $header, 'rows' => $rows)).
```

?>

Además, la función theme() puede tomar un arreglo de hooks theme, que pueden ser usados para proporcionar más específico control de la forma como Drupal presenta la información.

Es necesario que cada hook theme sea registrado por el módulo que lo declara. De esa forma, Drupal puede llamarlo y hacer todo lo que indica, haciendo fácil además identificarlo y sobreescribir el comportamiento que implementa.

Estas funciones hooks son registrados vía la función hook_theme que devuelve un arreglo de arreglos con información sobre el hook. Describe los argumentos que la función ó plantilla necesitará y proporciona funciones por defecto en el caso que no los encuentra. Si la implementación por defecto es una función, por convención es nombrada theme_HOOK().

Cada módulo debería proporcionar una implementación por defecto para todas las funciones theme_hooks que está registrando. Esta implementación puede ser en una función ó en una plantilla. Si es una función debe ser específicada vía la función hook_theme. Por convención, las implementaciones por defecto de los hooks theme son nombrados theme_HOOK. Las implementaciones de la plantilla por defecto son almacenados en el folder del módulo.

El renderizador por defecto de plantillas de Drupal es un simple motor PHP que incluye la plantilla y almacena la salida. El motor de temas Drupal puede proporcionar motores de plantilla alternativos tales como Xtemplate, Smarty y PHPTal. El motor de plantillas más común es PHPTemplate (incluído con Drupal e implementado en el archivo phptemplate.engine) que usa el renderizador por defecto que viene con Drupal.

Para crear implementaciones de temas específicos para estos hooks, los temas pueden implementar su propia versión de theme hooks.

4.3. La librería Content Construction Kit (CCK)

El Kit de Construcción de Contenidos (CCK) incluye métodos para que los desarrolladores PHP puedan usar y crear campos fuera de la interfase de usuario. Esto crea mucho más flexibilidad pero requiere un buen conocimiento del lenguaje PHP así como estar familiarizado con el estilo de desarrollo de Drupal.

El real poder de CCK reside en el hecho de que es una arquitectura basada en plugins qte hace fácil agregar más tipos de campos. Cientos de módulos CCK han

sido desarrollados, por eso es muy probable que algún campo que se necesite ya esté desarrollado por la comunidad. En algunas raras ocasiones, sin embargo, será necesario escribir código personalizado para llenar una específica necesidad.

Además, la capacidad de agregar campos a los tipos de contenidos a través del navegador es mucho más simple que tener que programar algo cada vez que necesites un campo más.

CCK proporciona una librería para crear tipos de campos y un número de hooks para permitir la interacción con datos y definiciones de datos. Para usar esta capacidad uno debe estar muy familiarizado con PHP y SQL y los conocimientos básicos para crear un módulo en Drupal usando su sistema de hooks.

Hay diferentes formas de usar las librerías de CCK: para crear de manera automática nuevos elementos tales como tipos de campos, widgets, y formateadores, para crear de manera automática instancias de elementos existentes y adjuntarlos a tipos de contenidos y para manipular de manera automática la carga y el almacenamiento de nodos que contienen datos CCK.

Los campos CCK están compuestos de un campo (que describe la forma que los datos son almacenados en la base de datos), un widget (que describe la forma en que usuario interectuará con el campo a través del formulario) y un formateador (que describe como los datos serán mostrados). Por ejemplo, el núcleo de CCK proporciona tres campos de número por defecto: entero, decimal y flotante que almacena datos en esos formatos. Los campos tipo número usan 2 widgets, un widget de campo de texto (textfield), donde el usuario puede tipear un número (como un número de teléfono ó una dirección) y el widget opciones (optionwidgets) que proporciona una lista de valores permitidos que serán presentados al usuario como una lista drop-down, checkboxes, ó radio buttons (como una lista de presets de códigos postales). Los formateadores para el campo Número proporciona una variedad de formas para mostrar el número y definir el caracter decimal y el separador de miles.

El campo y el widget pueden proporcionar también reglas de validación y otras configuraciones. La operación de validación para el campo puede incluir la lista de valores permitidos ó el máximo ó mínimo valor permitido. La operación de validación para el widget puede incluir chequeo de los datos para evaluar si están en el formato requerido por el widget.

Algunas configuraciones son comunes a todos los campos y widgets. Los campos tienen configuraciones que indican si ellos son requeridos y si ellos permiten valores múltiples por ejemplo. Los widgets tienen settings para etiquetar el campo, ayudar a mostrar el texto de ayuda al usuario final ó definir el valor por defecto

cuando se crea un nuevo nodo.

Los formateadores tienen diferentes configuraciones para mostrarse en cada modo de visualización del nodo. Así tiene uno para el modo resumen (teaser), otro para el texto completo (fulltext) y otro para la visualización del campo en un feed RSS.

4.3.1. Creación de módulos CCK

Un módulo puede servir para definir uno o más tipos de campos, tipos de widgets y formateadores. Para crear más campos tipo CCK es necesario descargar el módulo http://drupal.org/project/cck , de esa forma su API estará disponible para crear más campos.

El API de CCK está disponible a partir de sus hooks. Los campos CCK se llaman a partir de la función _content_field_invoke() . Los widgets CCK son tipicamente llamados por CCK cuando crea los elementos de formulario del campo enla visualización del nodo usando hook_form_alter().

Todos los módulos deberia declarar hook_content_notify(). Este hook debería ser implementado dentro del hook_install(), hook_uninstall(), hook_enable() y hook_disable(), y es usado para notificar al núcleo de CCK cuando un campo ó un widget es agregado o removido de forma que pueda responder apropiadamente. Este hook generalmente es usado para permitir que el módulo CCK pueda remover y datos creados por este módulo cuando el módulo es desinstalado ó para marcar el campo y su widget como activo ó inactivo. La ubicación recomendada para el hook content_notify es dentro del archivo de extensión .install.

El gráfico 4.3 muestra los componentes de un campos CCK

a. Módulos que definen tipos de campos

Los módulos que definen tipos de campos deberían implementar el hook_field_info() , hook_field_settings(), hook_field() y hook_content_content_is_empty(). Opcionalmente, también podemos implementar el hook_content_generate() para generar datos de prueba. Estos hooks son proporcionados por CCK. Los nombres de los campos y los widgets son truncados a 32 caracteres en la base de datos y en arreglos internos, tales como content_fields().

```
**
* Implementation of hook_field_info().
*/
function MYMODULE_field_info() {
  return array(
```

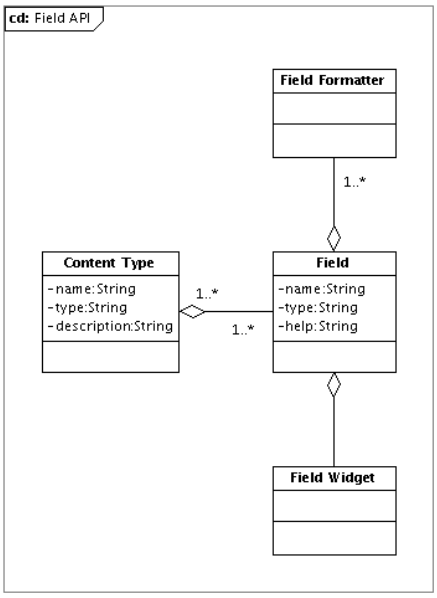

Figura 4.3: Componentes de un módulo CCK.
Fuente: [3]

```
    'example_field' => array(
      'label' => t('example_field'),
      'description' => t('This field will store example data.'),
      'content_icon' => 'icon_example.png',
    )
  );
}
```

El código para la función hook_field_settings() proporciona información al modulo CCK sobre que configuraciones estan disponibles para este tipo de campo y como las configuraciones deberían ser controladas por la sección “Manage fields” de la interfase de usuario. El valor que la variable $op tenga representa la operación que se efectuará. El valor para $field esta almacenada en la definición del arreglo. Los valores posibles para la operación son los siguientes:

- form construye los campos de formulario que servirán para almacenar los valores de configuración del campo.
- validate Valida si los datos ingresados en el formulario de configuración del campo tienen errores.
- save Proporciona un arreglo de las configuraciones que este campo usa y los que deberian ser almacenados en la definición del campo.
- database columns Declara las columnas que el módulo CCK creará y administra-

rá en representación del campo. Si el módulo campo desea administrar su propia base de datos entonces este valor deberá ser omitido. Los valores proporcionados deben escribirse usando Schema API.

- views data Proporciona datos al módulo Views. CCK llena valores por defecto automáticamente de modo que sólo se necesita implementa este valor si se desea sobrescribir la variable $data proporcionada por CCK ó si se desea administrar tu propio almacenamiento de base de datos

El siguiente código muestra la implementación por defecto de la función hook_field_settings() que incorpora estos valores.

```
/*
 * Implementation of hook_field_settings
 */

function MYMODULE_field_settings($op, $field) {
  switch ($op) {
          case 'form':
            $form = array();
            $form['max_length'] = array(
              '#type' => 'textfield',
              '#title' => t('Maximum length'),
              '#default_value' => is_numeric($field['max_length']) ?
                  $field['max_length'] : '',
              '#required' => FALSE,
              '#description' => t('The maximum length of the field.'),
            );
            return $form;
          case 'save':
            return array('max_length');
          case 'database columns':
            $columns['value'] = array(
              'type' => 'varchar',
              'length' => $field['max_length'],
              'not null' => FALSE,
              'sortable' => TRUE,
              'views' => TRUE,
            );
            return $columns;
          case 'views data':
            $allowed_values = content_allowed_values($field);
              if (count($allowed_values)) {
                        $data = content_views_field_views_data($field);
```

```
                $db_info = content_database_info($field);
                $table_alias = content_views_tablename($field);
            }
            //Argument : swap handler to the
            // 'many to one' operator.
            $data[$table_alias][$field['field_name'] .
            '_value']['argument']['handler'] =
            'content_handler_argument_many_to_one';
            return $data;
        }
}
```

El código en la función hook_field() define el comportamiento de un tipo de campo. Los valores son cargados en la variable $field. El valor $node es el que contiene los valores cargados en el objeto nodo. El valor de la variable $op definirá la operación a realizar. Los valores posibles para las operaciones son:

load El nodo acaba de ser cargado de la base de datos. Este hook debería ser usado para cargar el campo y debería devolver un objeto conteniendo valores extras para ser combinados junto con el objeto nodo.

validate El usuario acaba de editar el nodo y está intentando previsualizarlo ó almacenarlo. Este hook puede ser usado para chequear ó incluso para modificar el nodo. Los errores deberían ser configurados con la función form_set_error().

presave El usuario acaba de editar el nodo y el nodo ha pasado validación. Esta función puede ser usada para modificar el objeto nodo.

insert El nodo está siendo creado (insertado en la base de datos)

update El nodo está siendo actualizado

delete El nodo está siendo borrado

delete revision La versión revisión del nodo está siendo borrado.

sanitize El nodo está siendo visualizado, se necesita proporcionar un valor seguro para esto.

prepare translation Usado solamente por el campo Node Reference.

Las operaciones insert, update, delete, validate, y presave no devuelven valores. En muchos casos, solo las operaciones validate y sanitize son relevantes; el resto tienen implementaciones por defecto en la función content_field(). Es necesario implementar la variable $error_field para identificar donde se almacenará el error

en los elementos anidados. Este valor es pasado hacia la variable $element para que este disponible durante la validación. El código en la función hook_field() define el comportamiento de un tipo de campo. La implementación por defecto de esta función es como sigue:

```
/**
* Implementation of hook_field().
*/
function MYMODULE_field($op, &$node, $field, &$items, $teaser, $page) {
  switch ($op) {
    case 'validate':
      if (is_array($items)) {
        foreach ($items as $delta => $item) {
          $error_element = isset($item['_error_element']) ?
          $item['_error_element'] : '';
          if (is_array($item) &&
          isset($item['_error_element'])) unset($item['_error_element']);
          if (!empty($item['value'])) {
            if (!empty($field['max_length']) &&
             drupal_strlen($item['value']) > $field['max_length']) {
              form_set_error($error_element,
              t('%name: the value may not be longer than %max characters.
                  ',
              array('%name' => $field['widget']['label'],
              '%max' => $field['max_length'])));
            }
          }
        }
      }
      return $items;
    case 'sanitize':
      foreach ($items as $delta => $item) {
        $example = check_plain($item['value']);
        $items[$delta]['safe'] = $example;
      }
  }
}
```

b. Módulos que definen tipos de widgets

Los módulos que definen tipos de widgets necesitan implementar tres funciones: hook_widget_info(), hook_widget_settings(), y hook_widget(). Muchos de los campos CCK también implementan la función hook_elements() para sus widgets para permitir que que sean reutilizables y sean incorporados a través de #process.

En la función hook_widget_info(), nosotros definimos un nombre para el widget, especificando con que tipos de campos debería usarse este campo, e indicando si este módulo o el núcleo de CCK administrara los valores múltiples y el valor por defecto.

En las versiones anteriores de CCK, los widgets tenían que comprobar si un campo contenía valores múltiples con el número apropiado de valores. Este requerimiento ha cambiado en Drupal 6. Ahora el núcleo de módulo CCK administra esta tarea; pasa por el widget el número de veces que sea necesario. Este enfoque simplifica el código que se debe programar para implementar el widget aunque hay muchas veces donde tal comportamiento es no deseado. Si un módulo prefiere controlar directamente el número de valores del widget debe indicarlo en la función hook_widget_info().

Las siguientes constantes deben ser implementadas con valores múltiples

- CONTENT_HANDLE_CORE Este módulo espera que CCK gestione los valores múltiples
- CONTENT_HANDLE_MODULE Este módulo gestiona sus valores múltiples. Las siguientes configuraciones están disponibles para valores por defecto:
- CONTENT_CALLBACK_DEFAULT Este módulo espera que CCK gestione el valor por defecto.
- CONTENT_CALLBACK_CUSTOM Este módulo espera gestionar el valor por defecto de manera independiente.

Estas configuraciones de valores múltiples y valores por defecto pueden ser omitidos si se dejará toda la gestión al núcleo del CCK.

```
/**
* Implementation of hook_widget_info().
*/
function MYMODULE_widget_info() {
  return array(
    'example_widget' => array(
      'label' => t('Example widget'),
      'field types' => array('example'),
      'multiple values' => CONTENT_HANDLE_CORE,
      'callbacks' => array(
        'default value' => CONTENT_CALLBACK_DEFAULT,
      ),
    ),
```

```
  );
}
```

Los widgets también necesitan implementar hook_widgets_settings(). Esta función proporciona información al núcleo de CCK sobre que configuraciones estan disponibles para este widget y como estas configuraciones deberian ser gestionadas en la interface de usuario para gestión de campos. El valor de la variable $widget queda almacenada en un arreglo. Los valores posibles para la operaciones son como sigue:

form Se crea los elementos en la interface de configuración del campo.

validate Valida las configuraciones del widget, chequea para errores.

save Proporciona un arreglo de las configuraciones que el widget almacena en la definición del campo.

El siguiente código muestra la implementación del hook_widget_settings()

```
/**
* Implementation of hook_widget_settings().
*/
function example_widget_settings($op, $widget) {
  switch ($op) {
    case 'form':
      $form = array();
      $size = (isset($widget['size']) && is_numeric($widget['size'])) ?
       $widget['size'] : 60;
      $form['size'] = array(
        '#type' => 'textfield',
        '#title' => t('Size of textfield'),
        '#default_value' => $size,
        '#element_validate' =>
        array('_element_validate_integer_positive'),
        '#required' => TRUE,
      );
      return $form;
    case 'save':
      return array('size');
  }
}
```

Nuestra función hook_widget() recibe una referencia a las variables $form, $form_state, el arreglo $field, el arreglo $items que contiene los valores para este campo en el

actual nodo y la variable $delta que contiene el número de veces que se repite el widget en el formulario cuando se trabajan con valores múltiples.

```
/**
* Implementation of hook_widget().
*/
function MYMODULE_widget(&$form, &$form_state, $field,
$items, $delta = 0) {

  $element['value'] = array(
    '#type' => 'textfield',
    '#default_value' => isset($items[$delta]['value']) ?
    $items[$delta]['value'] : NULL,
    '#autocomplete_path' => $element['#autocomplete_path'],
    '#size' => !empty($field['widget']['size']) ?
    $field['widget']['size'] : 60,
    '#attributes' => array('class' => 'example'),
    '#maxlength' => !empty($field['max_length']) ?
    $field['max_length'] : NULL,
  );
  if (empty($form['#parents'])) {
    $form['#parents'] = array();
  }
  $element['_error_element'] = array(
    '#type' => 'value',
    '#value' => implode('][',
    array_merge($form['#parents'], array('value'))),
  );

  return $element;
}
```

c. Módulos que definen tipos de formateadores

Los formateadores de CCK definen como será visualizado el contenido del campo en los nodos y las vistas. Cada formateador define un arreglo de uno ó más tipos de campos con las cuales se puede usar. Muchos campos tienen diferentes formateadores disponibles que tu puedes escoger. Cada campo debería definir al menos un formateador.

En la interface de usuario, el formateador determina si un campo es oculto, es mostrado alineado horizontalmente ó debajo de la etiqueta.

La operación 'view' (controlada por el núcleo de CCK) construye el objeto $node de forma que también se puede usar drupal_render() para visualizar la salida forma-

teada para un campo individual. Los valores de campo serán renderizados hacia HTML solamente en el último momento para permitir que otros módulos puedan hacer cambios en los datos antes de que sean renderizados.

Después, cuando el nodo ha sido preparado para la visualización, la siguiente función renderizará completamente todo el campo.

```
print drupal_render($node->field_foo)
```

La función encargada de procesar un campo lo hace sólo una vez para el caso de un campo individual o pasa varias veces en el caso de un campo multiple. Los formateadores de campo múltiple pueden usarse para visualizar campos en un único mapa ó visualizar varios de ellos en un grafo. Ninguno de los módulos CCK del núcleo proporciona formateadores para valores múltiples, pero la capacidad es ofrecida de modo que los módulos contribuidos puedan implementarla si lo desean.

Los formateadores de valores únicos están implementados por defecto. Los formateadores de valores múltiples puede ser implementados usando la función formatter_info()

CONTENT_HANDLE_CORE Este indica que el formateador trabajará con un solo valor gestionado por el núcleo de CCK.

CONTENT_HANDLE_MODULE Indica que el formateador trabajará con valores múltiples y será gestionado por el módulo contribuido sin la intervención del núcleo de CCK.

La información que necesita el formateador va dentro de la función hook_formatter_info()

```
/**
* Implementation of hook_field_formatter_info().
*/
function example_field_formatter_info() {
  return array(
    'default' => array(
      'label' => t('Default'),
      'field types' => array('example'),
      'multiple values' => CONTENT_HANDLE_CORE,
    ),
    'plain' => array(
      'label' => t('Plain text'),
      'field types' => array('example'),
      'multiple values' => CONTENT_HANDLE_CORE,
```

```
    ),
  );
}
```

Si el formateador necesita crear elementos de theming, tu módulo debe declarar hook_theme() para cada formateador que se haya implementado. El núcleo de CCK una función con un nombre como

```
[MODULE_NAME]_formatter_[FORMATTER_NAME]
```

que recibirá un único argumento con valores a formatear. Un formateador de valores únicos recibirá el valor apropiado de la variable $delta embebido en los valores del campo dentro del arreglo del nodo. En cambio, un formateador de valores múltiples recibirá todos los ítems del subarray del contenido del nodo.

Una de las más importantes responsabilidades del formateador es renderizar una versión segura del contenido del campo (modo safe). La implementación por defecto de la función hook_theme es incluida debajo:

```
/**
* Implementation of hook_theme().
*/
function MYMODULE_theme() {
  return array(
    'example_formatter_default' => array(
      'arguments' => array('element' => NULL),
    ),
    'example_formatter_plain' => array(
      'arguments' => array('element' => NULL),
    ),
  );
}

/**
* Theme function for 'default' example field formatter.
*/
function theme_example_formatter_default($element) {
  return $element['#item']['safe'];
}

/**
* Theme function for 'plain' example field formatter.
*/
function theme_example_formatter_plain($element) {
  return strip_tags($element['#item']['safe']);
```

}

4.4. El ciclo de vida de una aplicación desarrollada en Drupal

Un buen plan de proyecto para Drupal comienza con el cliente. Un proyecto realizado con Drupal tiene 6 fases:

Descubrimiento

Durante el descubrimiento, nosotros aprendemos todo lo que debemos saber sobre el cliente y los objetivos del proyecto. En esta fase es donde comenzamos a crear un mapa de funcionalidades que necesitamos implementar, todos los recursos que nosotros necesitaremos, etc.

Experiencia de usuario y arquitectura

En esta fase se identifica el perfil de las personas que interactuarán con el proyecto, por ejemplo los usuarios finales y las personas que gestionarán la web cuando el proyecto esté terminado. Durante esta fase, se desarrrollan wireframes, flujos de usuario y frecuentemente comenzar a construir algunos prototipos en Drupal.

Prototipos

Durante esta fase que usualmente va antes de comenzar la fase de implementaciones funcionales, se comienza a probar algunas de las hipótesis y flujos de usuario que vienen de la fase de experiencia de usuario. Para sitios simples, esta fase y la fase de implementación funcional van juntos, en cambio, para fujos de usuarios más complejos, ó para proyectos donde se va a manejar mucho contenido, la fase de prototipado es esencial para estar seguro de que la implementación que vas a construir funcionará en Drupal.

Implementación funcional

Durante esta fase, el objetivo es crear la funcionalidad que se ha descrito en la fase de experiencia de usuario y allanar cualquier área donde la funcionalidad que hemos decidido no tiene sentido, ó cae fuera del presupuesto. Para sitios pequeños, es una buena oportunidad para desarrollar tu experiencia como desarrollador. Si el sitio web es complejo entonces es mejor contar con un equipo de desarrolladores que puedan solucionar implementaciones complejas. Los desarrolladores son el mejor amigo de un diseñador Drupal.

Diseño visual y theming

El diseño visual (colores, fuentes de letras, imágenes y otros elementos de marcas) recién se define aquí. Hay muchas razones para esto, la principal es que trabajar el diseño visual al inicio del proyecto es una receta para el desastre. Parte del trabajo del diseñador de Drupal es ayudar al cliente a mantenerse enfocado en lo más importante, es decir, en como el sitio web servirá a los objetivos del proyecto y a su marca. Mientras el diseño visual es un componente importante del valor del sitio, es sólo una pieza de él. Hay cuestiones más importantes como decidir si el visitante necesita 50 páginas de contenido al momento de hacer una compra online. La mejor forma de explicar esto a los clientes es que la primera parte del proceso es para recrear la experiencia de usuario. La fase de diseño/theming se asegura que la experiencia que has diseñado en las fases anteriores mezcla la marca del cliente con el mensaje que se quiere transmitir.

Prueba y lanzamiento

Siempre se prueba antes y después de lanzar. Hay unos pasos que seguir: primero, mover el codigo del servidor de desarrollo al servidor de ensayo y asegurarse que todas las funcionalidades están trabajando bien. Una vez que todo está comprobado que trabaja bien, se traslada el código del servidor de ensayo al servidor de producción.

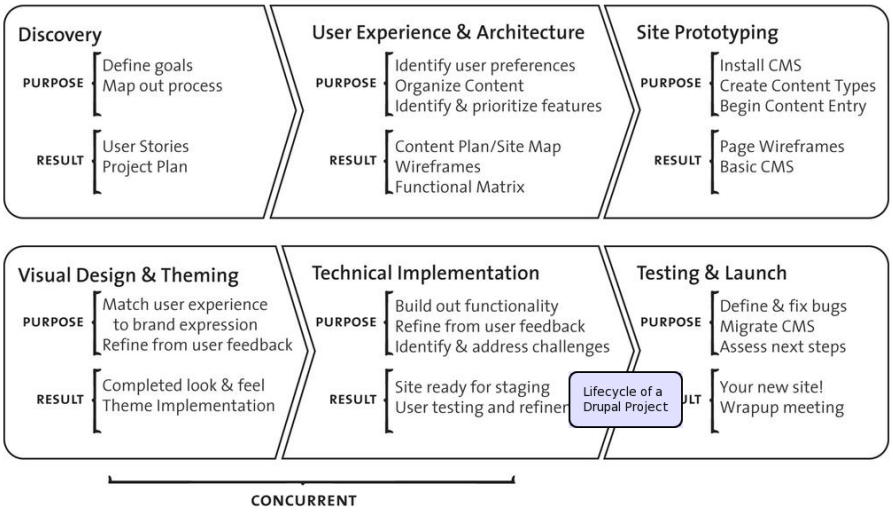

Figura 4.4: Ciclo de vida de una aplicación construida en Drupal. Fuente: [5]

CAPÍTULO 5

IMPLEMENTACIÓN DE LA APLICACIÓN

5.1. Análisis de Requerimientos

5.1.1. Requerimientos funcionales

a. La aplicación deberá extraer los términos actualizados de la base de datos AGROVOC.

b. La aplicación debe permitir seleccionar términos provenientes del tesauro AGROVOC de forma amigable.

c. La aplicación deberá poder permitir configurar los parámetros para la recuperación remota de términos provenientes del AGROVOC.

d. La aplicación deberá permitir configurar la visualización de los datos al usuario en la vista de texto completo y en la vista de listado.

e. La aplicación debe generar automáticamente términos de un archivo subido.

f. La aplicación debe ser modular.

g. La aplicación debe ser fácil de instalar.

h. La aplicación debe generar automáticamente versiones en los idiomas disponibles en el sitio web por cada término seleccionado.

i. La aplicación debera poder asociar el término seleccionado con el URI que le corresponde.

5.1.2. Requerimientos no funcionales

a. Sus componentes deben estar basados en software libre.

b. Estará instalado en una infraestructura LAMP (Linux, Apache, MySQL y PHP)

c. La aplicación debe correr en conexión con internet.

d. La aplicación debe desarrollarse bajo control de versiones.

e. Deberá usar un framework soportado por una comunidad amplia de software libre.

5.2. Diseño y Arquitectura

En base a los requerimientos establecidos en la sección anterior se eligió a Drupal como framework para el desarrollo de la aplicación. Drupal tiene una arquitectura modular y está soportada por una gran comunidad de usuarios y desarrolladores.

Drupal permite implementar sistemas de gestión de contenidos robustos gracias a que provee al desarrollador de un API para desarrollar todos los elementos que necesita para implementar este tipo de sistemas como: tipos de contenidos, vistas, campos y taxonomías. La aplicación que vamos a desarrollar consistirá en una entidad denominada campo agrovocfield y dos widgets (autocomplete y automatic indexing) para visualizar la información almacenada en esa entidad. La aplicación que se va a desarrollar está ubicada en las dos capas superiores del siguiente esquema y depende de la instalación del core de Drupal y de los módulos CCK, Agrovoc_API, i18n y taxonomy.

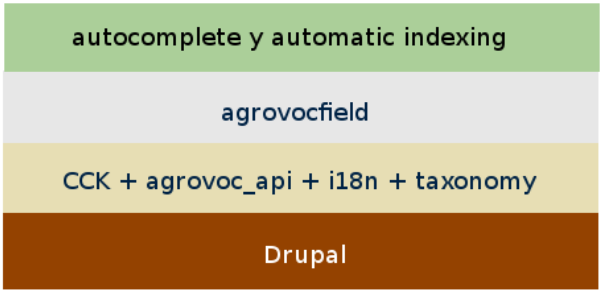

Figura 5.1: Arquitectura del módulo agrovocfield.
Fuente: Elaboración propia

La aplicación a desarrollar se implementará bajo este ambiente ya que puede permitirle ser reusado por cualquier sistema que use Drupal. En términos concretos la aplicación permitirá instalar un campo que permita seleccionar los términos provenientes del tesauro AGROVOC. El campo podrá ser usado en cualquier tipo de contenido construído en Drupal y los parámetros de uso podrán ser configurados en la ventana de configuración del campo.

La ventana de configuración permitirá elegir entre dos widgets, uno que sugiere automáticamente términos de AGROVOC mientras escribe el usuario (autocomplete widget) y otro que sugiere términos extraídos automáticamente de un archivo (automatic indexing).

La figura 5.2 muestra cómo se realiza el proceso de recuperación de términos desde la base de AGROVOC para el widget denominado Autocomplete.

La figura 5.3 muestra cómo se realiza el proceso de recuperación de términos desde la base de AGROVOC para el widget denominado Automatic Indexing :

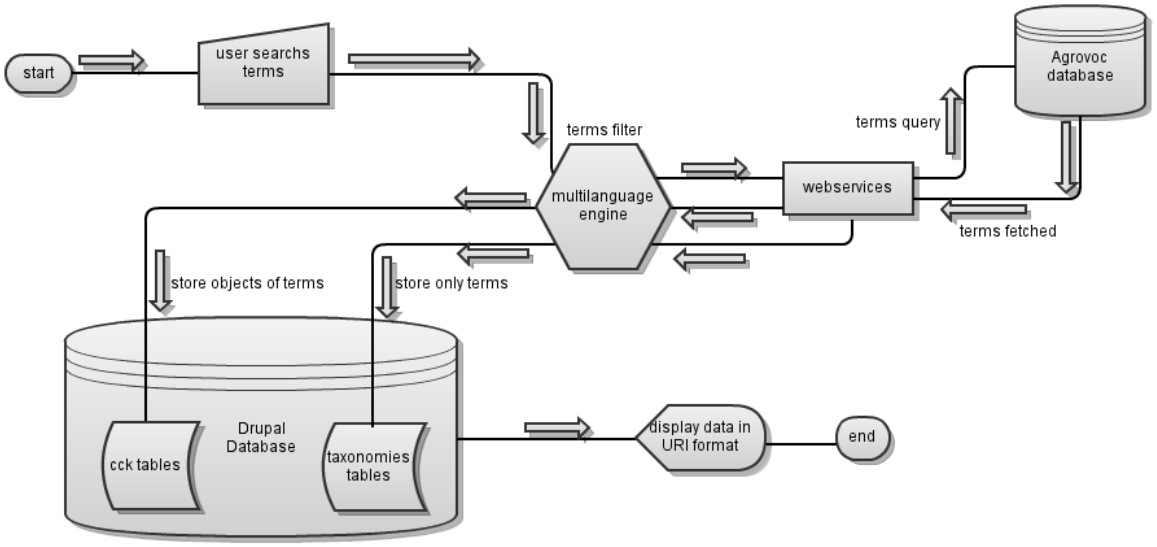

Figura 5.2: Diagrama de flujo para el widget Autocomplete.
Fuente: Elaboración propia

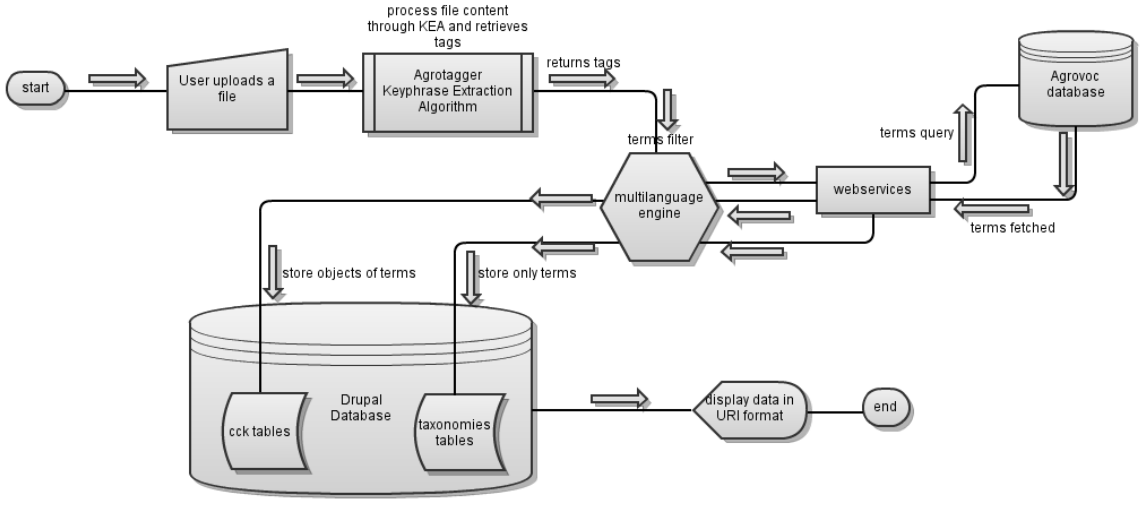

Figura 5.3: Diagrama de flujo para el widget Automatic Indexing.
Fuente: Elaboración propia

El proceso de extracción de términos de un archivo usando el algoritmo KEA se ve representado en la figura 5.4:

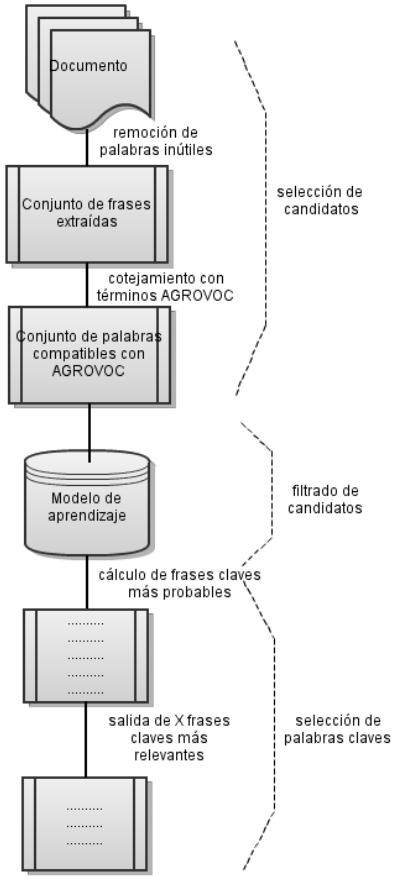

Figura 5.4: Proceso de extracción de términos usando el algoritmo KEA.
Fuente: Elaboración propia

5.2.1. Diagramas de Secuencia

El diagrama de secuencia del widget Autocomplete se muestra en la figura 5.5

El diagrama de secuencia del widget Automatic Indexing se muestra en la figura 5.6

5.2.2. Diagrama de Componentes

El diagrama de Componentes se muestra en la figura 5.7

5.2.3. Diagrama de Despliegue

El diagrama de Despliegue se muestra en la figura 5.8

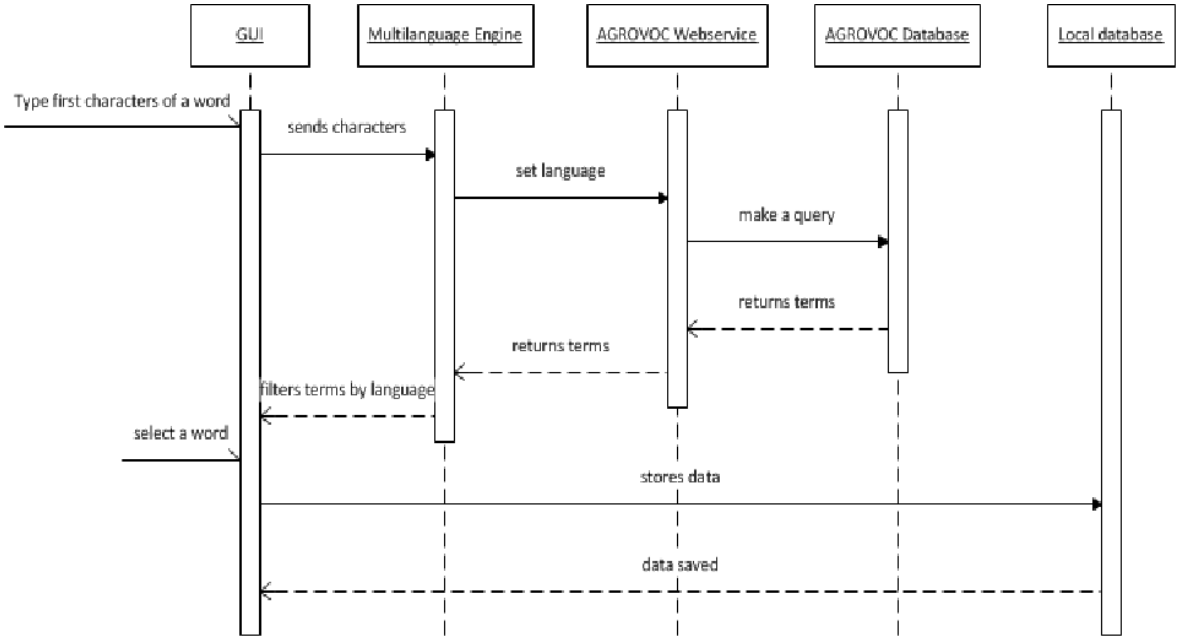

Figura 5.5: Diagrama de Secuencia del Widget Autocomplete

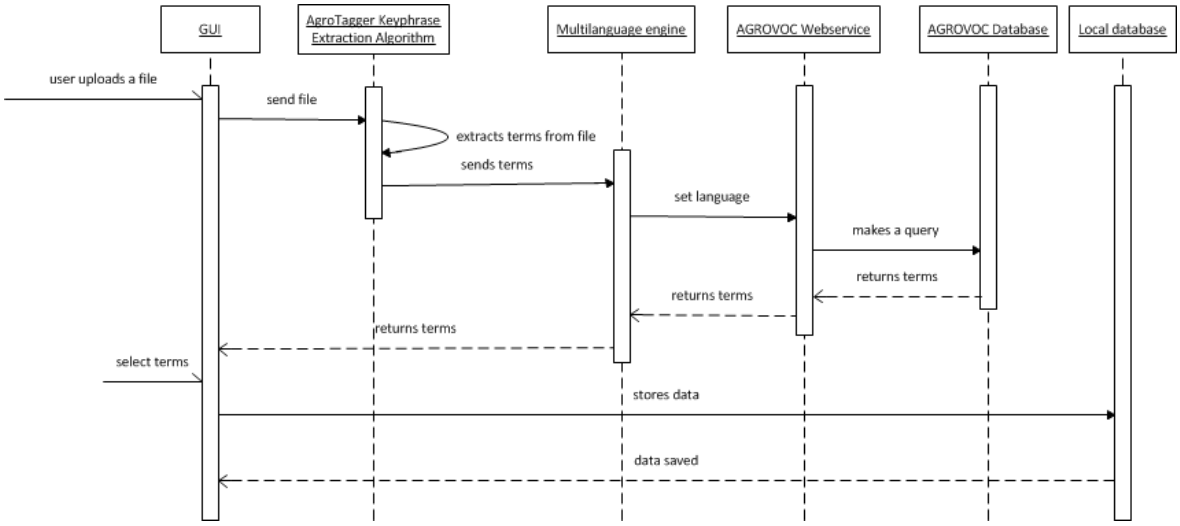

Figura 5.6: Diagrama de Secuencia del Widget Automatic Indexing

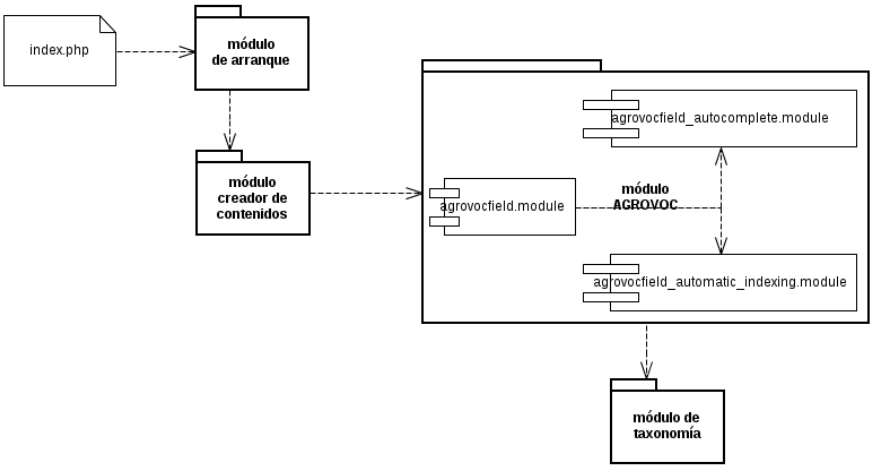

Figura 5.7: Diagrama de Componentes

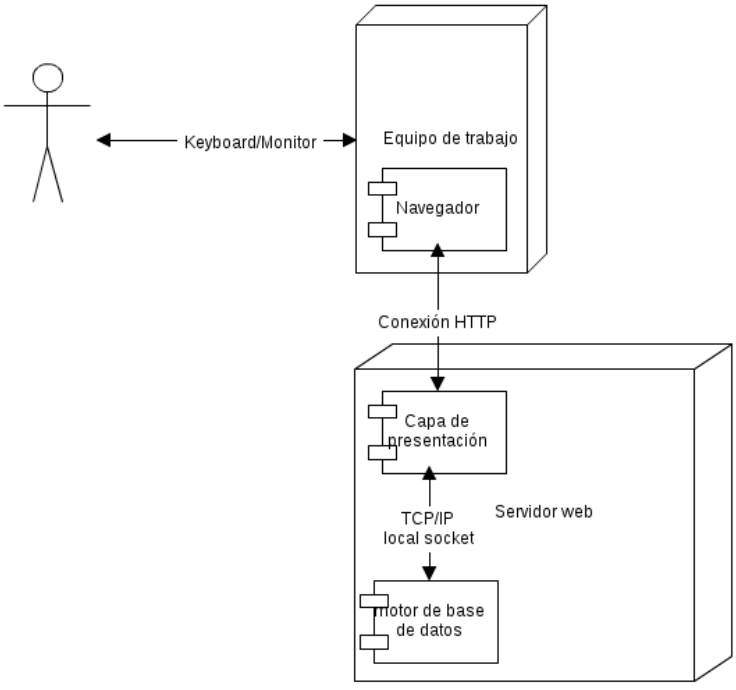

Figura 5.8: Diagrama de Despliegue

5.2.4. Diagrama Entidad Relación

La estructura de las tablas que almacenarán la información se muestra en la gráfica 5.9:

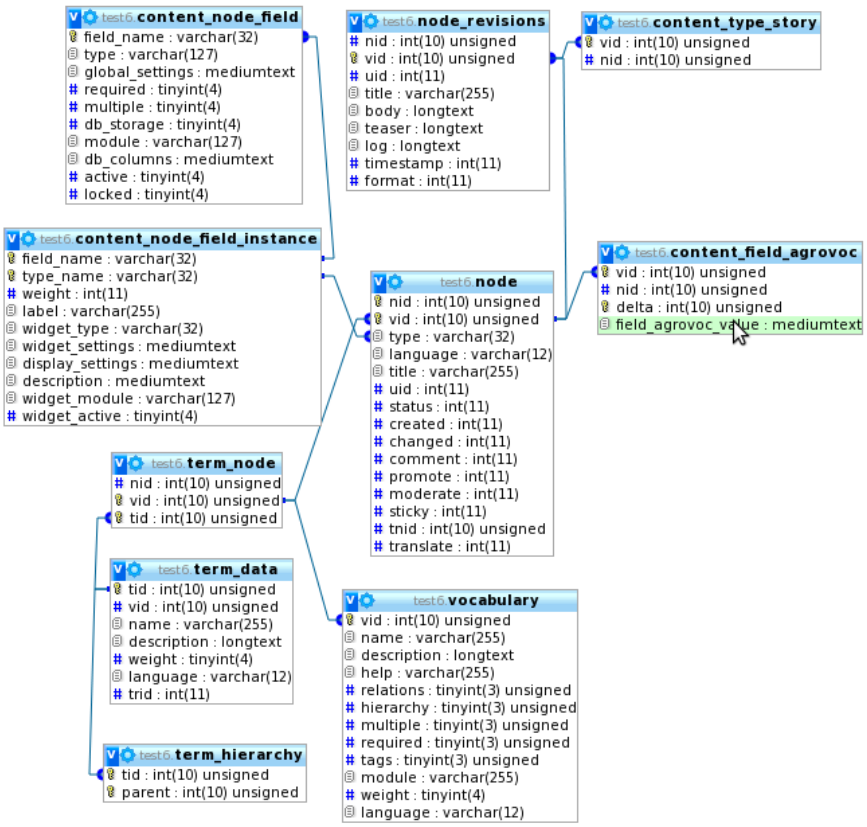

Figura 5.9: Diagrama entidad - relación para la aplicación Agrovocfield.
Fuente: Elaboración propia

5.2.5. Interfaz gráfica de usuario para el widget Autocomplete

a. Ventana de configuración del widget

   Se puede ver en la figura 5.10

b. Ventana de configuración de la visualización del widget

   Se puede ver en la figura 5.11

c. Ventana de visualización del widget autocomplete

   Se puede ver en la figura 5.12

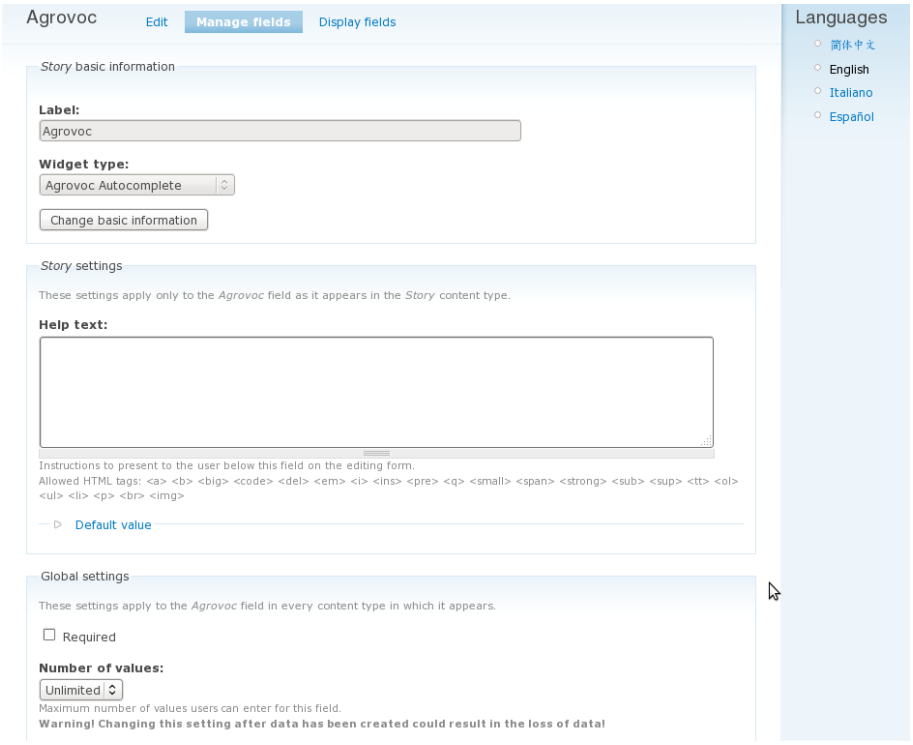

Figura 5.10: Interfaz gráfica de usuario para el widget autocomplete.
Fuente: Elaboración propia

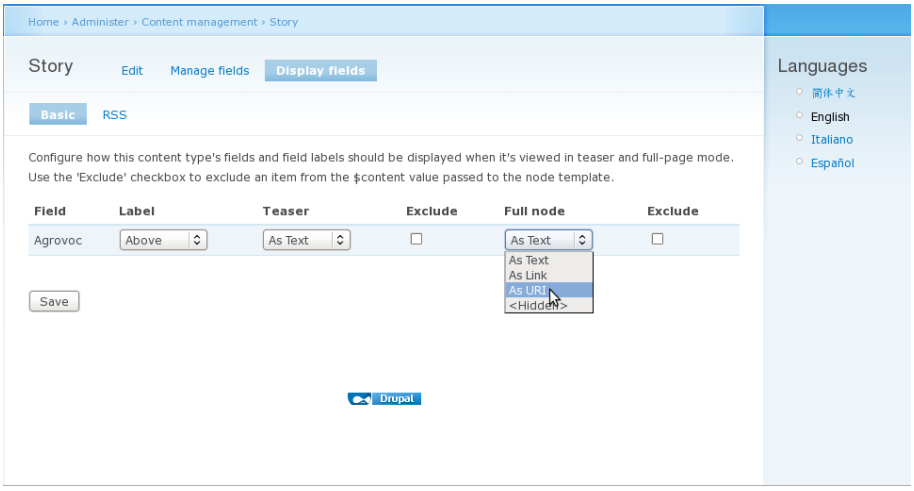

Figura 5.11: Ventana de configuración del modo de visualización del widget autocomplete.
Fuente: Elaboración propia

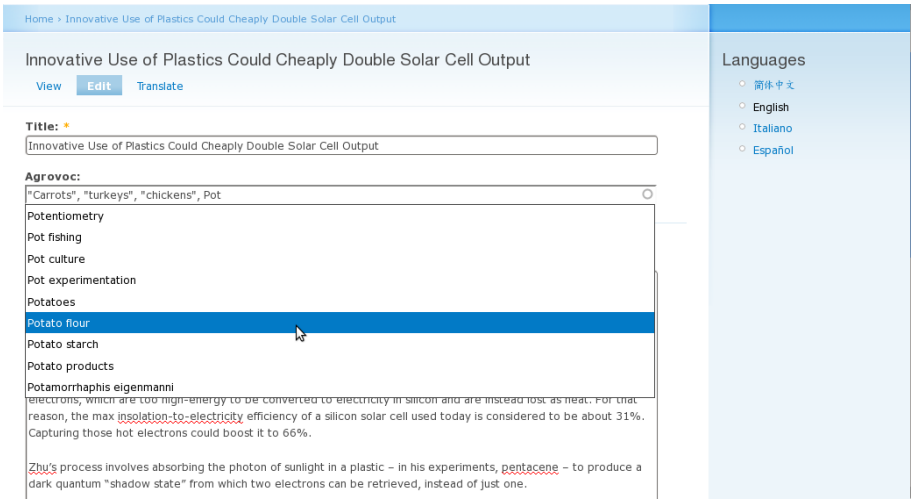

Figura 5.12: Interfaz gráfica que muestra como funciona el widget autocomplete.
Fuente: Elaboración propia

5.2.6. Interfaz gráfica de usuario para el widget Automatic Indexing

a. Ventana de configuración del widget

   Se puede ver en la figura 5.13

b. Ventana de uso del widget

   Se puede ver en la figura 5.14

5.2.7. Interfaz gráfica de usuario para el almacenamiento de los términos

Se puede ver en la figura 5.15

5.2.8. Interfaz gráfica de usuario para la visualización de los términos en diferentes idiomas

a. Visualización de los términos en inglés

   Se puede visualizar en la figura 5.16

b. Visualización de los términos en español

   Se puede visualizar en la figura 5.17

c. Visualización de los términos en chino

   Se puede visualizar en la figura 5.18

d. Visualización de los términos en italiano

   Se puede visualizar en la figura 5.19

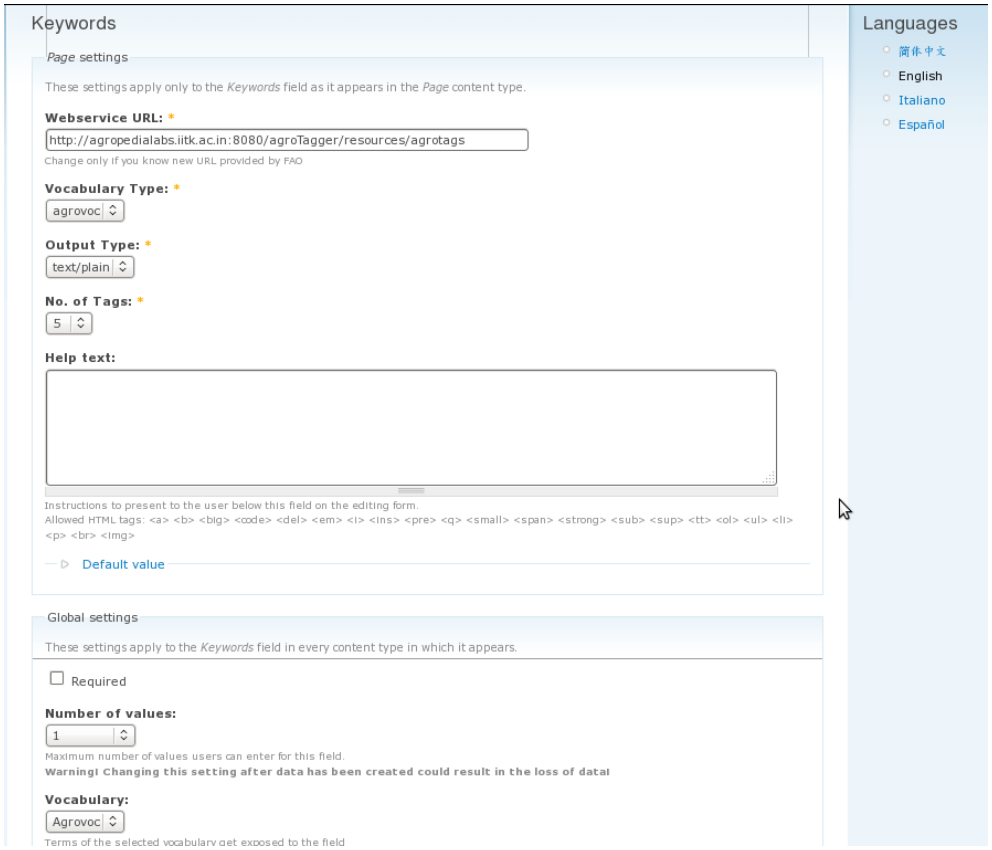

Figura 5.13: Interfaz gráfica que muestra la ventana de configuración del widget automatic indexing.
Fuente: Elaboración propia

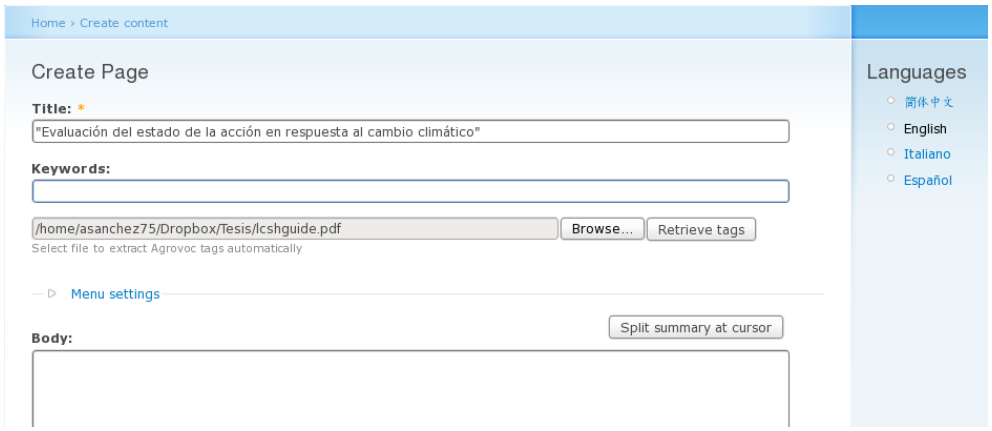

Figura 5.14: Interfaz gráfica que muestra como funciona el widget autocomplete.
Fuente: Elaboración propia

Figura 5.15: Interfaz gráfica de usuario para el almacenamiento de los términos.
Fuente: Elaboración propia

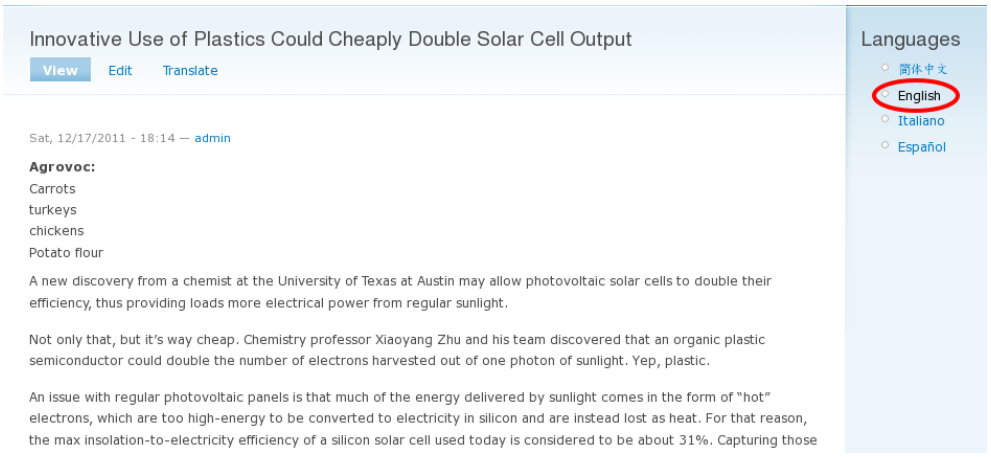

Figura 5.16: Visualización de los términos en inglés.
Fuente: Elaboración propia

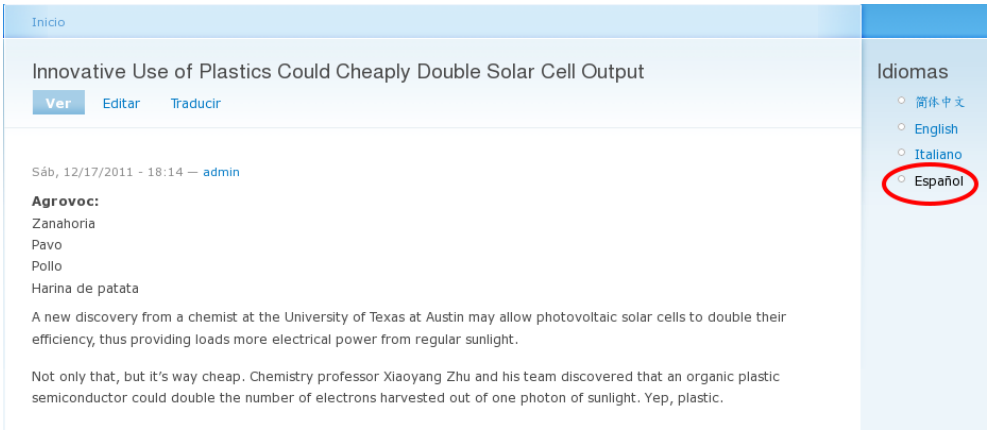

Figura 5.17: Visualización de los términos en español.
Fuente: Elaboración propia

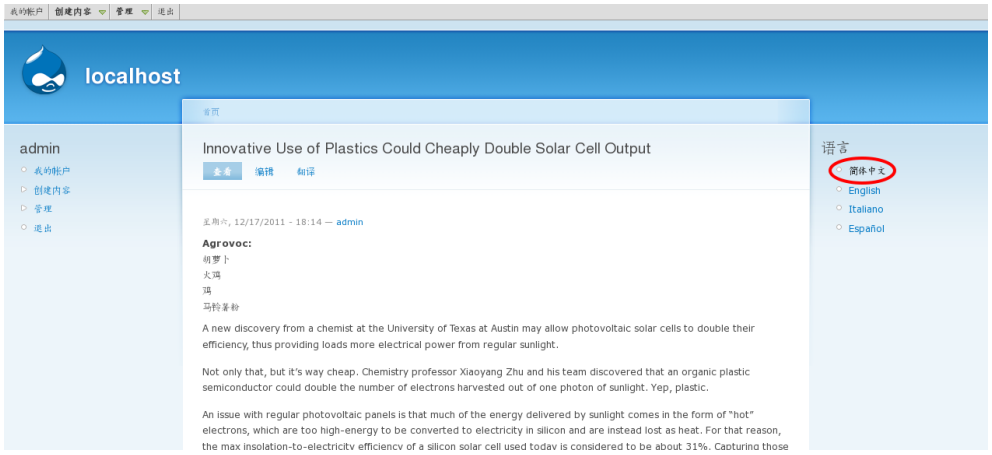

Figura 5.18: Visualización de los términos en chino.
Fuente: Elaboración propia

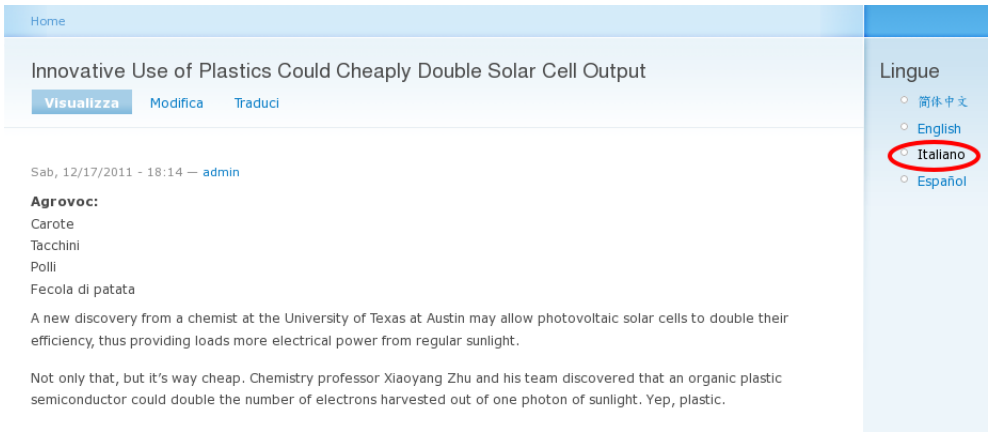

Figura 5.19: Visualización de los términos en italiano.
Fuente: Elaboración propia

5.2.9. Interfaz gráfica de usuario para la visualización de los URIs en el servidor de la FAO

Se puede visualizar en la figura 5.20

Figura 5.20: Visualización de los URIs en la página web de la FAO.
Fuente: Elaboración propia

5.3. Programación

5.3.1. Pseudocódigo para el widget Autocomplete

Procedure Autocomplete(T)

```
// El sistema sugiere términos al usuario mientras va tipeando;
```

foreach *x in T* **do**

leer(x)

```
// envía consulta al servidor remoto donde está AGROVOC;
```

consultar(x)

```
// recibe un objeto con las traducciones del término y su
identificador en AGROVOC;
```

recibir(arreglo [x])

```
// almacena todas las traducciones del término y su
identificador;
```

almacena(arreglo [x])

end

End procedure

El diagrama de flujo del pseudocódigo se puede ver en la figura 5.21

5.3.2. Pseudocódigo para el widget Automatic Indexing

```
Procedure Automatic Indexing(F)
//El usuario sube el archivo;
Subir archivo(x)
foreach archivo in F do
    Enviar(x)
    //una aplicación remota procesa el archivo y devuelve las
    palabras claves;
    Recibir([[x]])
    foreach x in T do
        leer(x)
        //envía consulta al servidor remoto donde está AGROVOC;
        consultar(x)
        //recibe un objeto con las traducciones del término y su
        identificador en AGROVOC;
        recibir(arreglo [x])
        //almacena todas las traducciones del término y su
        identificador;
        almacena(arreglo [x])
    end
end
End procedure
```

El diagrama de flujo del pseudocódigo se puede ver en la figura 5.22

5.3.3. Pruebas

a. Para el widget Autocomplete

La aplicación se está usando en el proyecto Global Rangelands `http://globalrangelands.org/` de la universidad de Arizona en USA y en el proyecto AgriDrupal auspiciado por la FAO `http://aims.fao.org/tools/agridrupal`.

b. Para el widget Automatic Indexing

En la página principal del proyecto KEA se puede ver unos ejemplos que muestran la calidad de la indización automática frente a la indización manual.

- Ejemplo 1

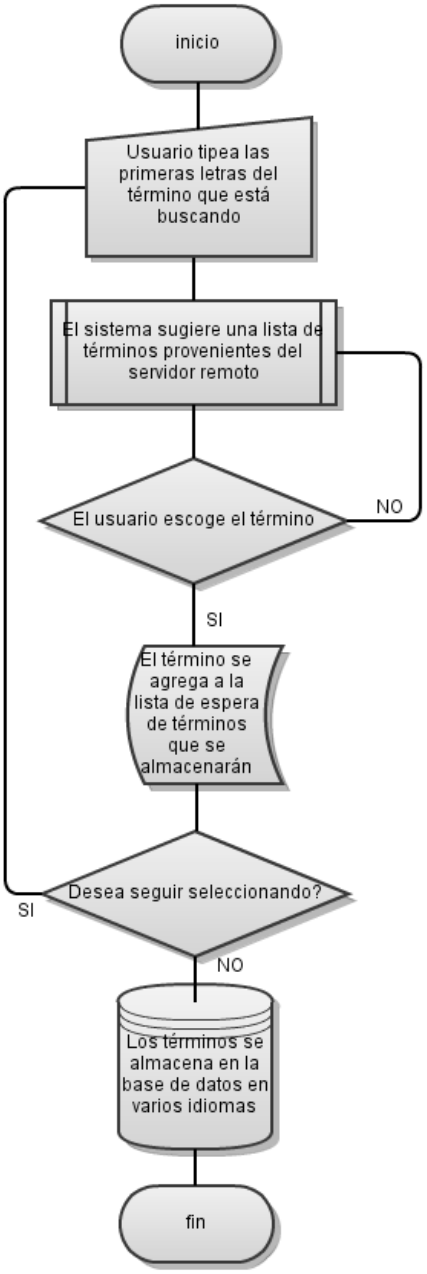

Figura 5.21: Diagrama de flujo para el widget autocomplete.
Fuente: Elaboración propia

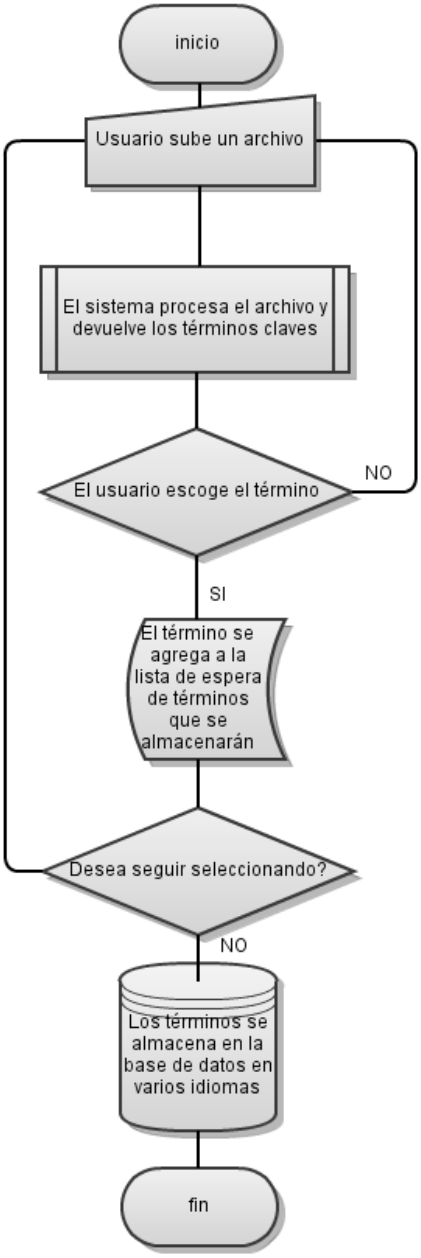

Figura 5.22: Diagrama de flujo para el widget automatic indexing.
Fuente: Elaboración propia

Se comparan los resultados de la indización automática frente a la indización manual del documento “The growing global obesity problem: Some policy options to address it” . La indización manual fue realizada por 6 indizadores profesionales y son mostradas en círculos coloreados mientras que los términos indizados por el algoritmo KEA son mostrados en círculos de color verde. Ver figura 5.23

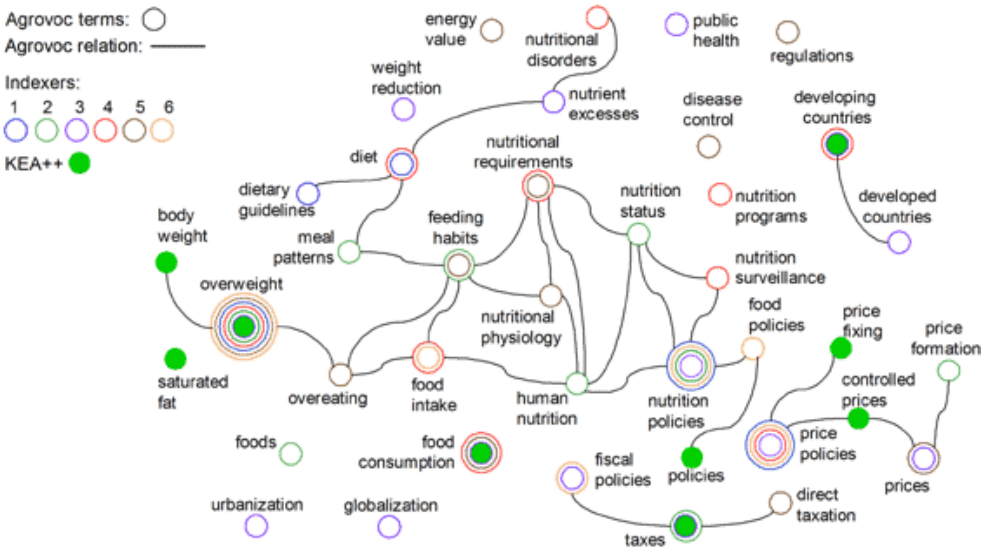

Figura 5.23: Indización manual vs Indización automática.
Fuente: Sitio web KEA

- Ejemplo 2

  Se muestran los resultados de la indización automática y manual para tres documentos. Los resultados muestran la confiabilidad de la indización automática.

The growing global obesity problem: Some policy options to address it		Overview of techniques for reducing bird predation at aquaculture facilities		Feeding Asian cities: Food production and processing issues	
Indexers	KEA	Indexers	KEA	Indexers	KEA
developing countries, food consumption, overweight, taxes, prices, price policies, fiscal policies, feeding habits, nutritional requirements, diet, nutrition policies, food intake	developing countries, food consumption, overweight, taxes, price fixing, controlled prices, policies, body weight, saturated fats	aquaculture, damage,fencing, noise,scares, bird control, fishery production, predatory birds, control methods, fish culture, noxious birds	aquaculture, damage, fencing, noise, scares, birds, fishing operations, predators, ropes	food policies, food supply, towns, urbanization, urban agriculture, urban areas, food production, agricultural policies, rural urban relations, asia, state intervention, agricultural sector, suburban agriculture	food policies, food supply, towns, urbanization, urban agriculture, urban areas, food consumption, new products

Cuadro 5.1: Cuadro Comparativo entre indización automática e indización manual. Fuente: Sitio web KEA

CONCLUSIONES

a. La sostenibilidad de una aplicación web depende de elegir un framework de gestión de contenidos soportado por una comunidad amplia de usuarios y desarrolladores. La aplicación está alojada en `http://drupal.org/project/agrovocfield`.

b. La indización automática mediante extracción de términos de un archivo puede ser muy confiable si se entrena bien al sistema y se cuenta con un tesauro con términos consistentes.

c. La indización usando el widget autocomplete está sujeta a la habilidad del indizador para la selección de los términos a pesar de que la aplicación le facilita el trabajo.

d. El uso de servicios web para sugerir términos en tiempo real durante el proceso de indización permite que se cuente de manera confiable con términos consistentes.

e. Es posible reusar la aplicación con otros tesauros temáticos siempre y cuando los webservices implementen las mismas funciones de consulta. Si no fuese el caso, es posible reeditar las funciones de la aplicación dado que está basado en licencia GNU.

f. Los términos extraídos automáticamente de los documentos están relacionados con agricultura ya son parte del dominio del tesauro AGROVOC. Si se desea indizar documentos de otra índole temática se tiene que entrenar al sistema con los tesauros respectivos.

RECOMENDACIONES

a. Se recomienda la actualización de la aplicación a Drupal 7 para incrementar su adopción por la comunidad de software libre.

b. Desarrollar un widget para la selección jerárquica de términos.

c. Usar bases de datos replicables como CouchDB para evitar los problemas de conexión que aparecen cuando se cae el webservice.

d. Se recomienda entrenar al webservice que extrae los términos de un archivo en otros idiomas diferentes al inglés.

e. Se recomienda usar un mismo webservice para los dos servicios de extracción de términos de un archivo y consulta de versiones idiomática de un término.

BIBLIOGRAFÍA

[1] Murtha Baca. *Introduction to Metadata*. The Getty Research Institute, 2008.

[2] JOMC 50/EIS Research Initiative. *What's Library of Congress Subject Headings*. Published by web, 2011.

[3] Web Initiative. Building drupal with reusable fields, Diciembre 2010.

[4] Isidoro Gil Leiva. *La automatización de la indización, propuesta teórico- metodológica:aplicación al área de biblioteconomía y documentación*. Universidad de Murcia, 1997.

[5] Dany Nordin. *Planning and Managing Drupal Projects*. O'Reilly, 1997.

[6] Marek Obitko. Web ontology language owl, Diciembre 2011.

[7] Ioannis Papadakis. *Subject-based Information Retrieval within Digital Libraries Employing LCSHs*. Published by web, 2009.

[8] M.E. STEVENS. Automatic indexing: a state of the art report. (monograph 91).

[9] Ana María Martínez Tamayo. *Indización y Clasificación en Bibliotecas*. Alfagrama, 2008.

[10] Blanca Gil Urdiciain. *Manual de Lenguajes Documentales*. Noesis, 1996.

[11] John K. VanDyk. *Pro Drupal Development*. Apress, 2008.

[12] W3C. Owl web ontology language, Diciembre 2011.

[13] W3C. Rdf vocabulary description language 1.0: Rdf schema, Diciembre 2011.

[14] W3C. Resource description framework (rdf), Diciembre 2011.

[15] W3C. W3c semantic web frequently asked questions, Diciembre 2011.

[16] Marcia Zeng. *Metadata*. Neal Schuman Publisher Inc, 2008.

APÉNDICE A

WIDGET PARA BÚSQUEDA INCREMENTAL DE TÉRMINOS

Hello, here is some text without a meaning. This text should show, how a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like »Huardest gefburn«. Kjift – Never mind! A blind text like this gives you information about the selected font, how the letters are written and the impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for a special contents, but the length of words should match to the language. Programa en PHP para la implementación del widget de búsqueda incremental de términos.

```
Contenido de archivo agrovocfield.info
name = Agrovoc CCK
description = CCK field that fetched and stored terms from AGROVOC
    thesaurus in several languages.
dependencies[] = content
dependencies[] = agrovoc_api
dependencies[] = taxonomy
dependencies[] = i18n
dependencies[] = i18ntaxonomy
package = CCK
core = 6.x
Contenido de archivo agrovocfield.module
<?php
/**
 * @file
 *  Agrovoc CCK field
 *  This module provides an CCK field to fetch and store terms from
    AGROVOC thesaurus
 *  Some portions of code were taken from Content Taxonomy Module and
     adapted to our objectives.
 */

/*
 * Agrovoc Server URI
 */

define('AGROVOC_URI', 'http://aims.fao.org/aos/agrovoc/c_');
```

```
/**
 * Implementation of hook_field_info().
 */
function agrovocfield_field_info() {
  return array(
    'agrovocfield' => array(
      'label' => t('Agrovoc field'),
        'description' => t('CCK field for AGROVOC'),
        'callbacks' => array(
          'tables' => CONTENT_CALLBACK_DEFAULT,
          'arguments' => CONTENT_CALLBACK_DEFAULT,
        ),
      ),
    );
}

/**
 * Implementation of hook_field().
 */
function agrovocfield_field($op, &$node, $field, &$items, $teaser, $page)
    {
  switch ($op) {
    case 'presave':
       global $language;
          foreach ($items as $item) {
            if ($item['value']) {
            $obj = agrovocfield_get_tids_from_labels_object(unserialize(
                $item['value']), $field['#vid']);
            $tid = $obj->labels[$language->language]['tid'];
            if (is_array($node->taxonomy[$field['vid']])) {
              if (!in_array($tid, $node->taxonomy[$field['vid']])) {
                $node->taxonomy[$field['vid']][] = $tid;
              }
            }
            // when saving an existing node without presenting a form to
                the user,
            // the terms are objects keyed by tid. there's no need to re-
                set these
            // terms, and to do so causes php warnings because the
                database rejects
            // the row insert because of primary key constraints.
            else {
              $node->taxonomy[$field['vid']] = array($tid);
              }
```

```
            }
          }
          if (empty($node->taxonomy)) {
           $node->taxonomy[$field['vid']] = NULL;
          }

        break;
    }
}

/**
 * Implementation of hook_field_settings().
 */
function agrovocfield_field_settings($op, $field) {
  switch ($op) {
        case 'form':
          $options_voc = array();
        foreach (taxonomy_get_vocabularies() as $voc) {
          $options_voc[$voc->vid] = $voc->name;
        }
        $form['vid'] = array(
          '#title' => t('Vocabulary'),
          '#type' => 'select',
          '#default_value' => is_numeric($field['vid']) ? $field['vid'] :
             0,
          '#options' => $options_voc,
          '#description' => t('Terms of the selected vocabulary get exposed
              to the field'),
        );
        return $form;
      case 'save':
        return array('vid');
      case 'database columns':
        return array(
            'value' => array('type' => 'text', 'size' => 'medium', 'not
               null' => FALSE),
        );
    }
}

/**
 * Implementation of hook_theme().
 */
function agrovocfield_theme() {
```

```
  return array(
    'agrovocfield_formatter_default' => array(
      'arguments' => array('element' => NULL),
    ),
    'agrovocfield_formatter_link' => array(
      'arguments' => array('element' => NULL),
    ),
    'agrovocfield_formatter_uri' => array(
      'arguments' => array('element' => NULL),
    ),
  );
}

/**
 * Implementation of hook_field_formatter_info().
 */
function agrovocfield_field_formatter_info() {
  return array(
    'default' => array(
      'label' => t('As Text'),
      'field types' => array('agrovocfield'),
      'multiple values' => CONTENT_HANDLE_CORE,
    ),
    'link' => array(
      'label' => t('As Link'),
      'field types' => array('agrovocfield'),
      'multiple values' => CONTENT_HANDLE_CORE,
    ),
    'uri' => array(
      'label' => t('As URI'),
      'field types' => array('agrovocfield'),
      'multiple values' => CONTENT_HANDLE_CORE,
    ),
  );
}


/**
 * Theme function for 'default' text field formatter.
 */
function theme_agrovocfield_formatter_default($element) {
    if (!empty($element['#item']['value'])) {
      global $language;
      $obj = unserialize($element['#item']['value']);
```

```
      return check_plain(agrovocfield_set_label_lang($obj, $language->
          language));
    }
}

/**
 * Theme function for 'link' text field formatter.
 */
function theme_agrovocfield_formatter_link($element) {
  if (!empty($element['#item']['value'])) {
    global $language;
    $field = content_fields($element['#field_name'], $element['#type_name
        ']);
    $term_object = unserialize($element['#item']['value']);
    $term = agrovocfield_get_tids_from_labels_object($term_object ,
        $field['vid']);
    $label = $term->labels[$language->language]['name'];
    $tid = $term->labels[$language->language]['tid'];
    return l($label, 'taxonomy/term/' . $tid, array('attributes' => array
        ('rel' => 'tag', 'title' => $label)));
  }
}

/**
 * Theme function for 'uri' text field formatter.
 */
function theme_agrovocfield_formatter_uri($element) {
  if (!empty($element['#item']['value'])) {
    global $language;
    $field = content_fields($element['#field_name'], $element['#type_name
        ']);
    $term_object = unserialize($element['#item']['value']);
    $term = agrovocfield_get_tids_from_labels_object($term_object ,
        $field['vid']);
    $label = $term->labels[$language->language]['name'];
    $agrovoc_id      = $term->id;
    return l($label, AGROVOC_URI . $agrovoc_id, array('attributes' =>
        array('rel' => 'tag', 'title' => $label)));
  }
}

/**
 * Implementation of hook_content_is_empty().
 */
```

```
function agrovocfield_content_is_empty($item, $field) {
  if (empty($item['value'])) {
    return TRUE;
  }
  return FALSE;
}

/**
 * Adapted from taxonomy_get_term_by_name function.
 * I did it because I can't use hook_db_rewrite_sql to avoid filters by
    languages
 *
 * @param string $name
 *   Term searched
 * @return
 *   array of objects for each term found
 */
function agrovocfield_get_term_by_name($name) {
  $db_result = db_query("SELECT t.* FROM {term_data} t WHERE LOWER(t.name
     ) = LOWER('%s')", trim($name));
  $result = array();
  while ($term = db_fetch_object($db_result)) {
    $result[] = $term;
  }
  return $result;
}

/**
 * Adapted from taxonomy_get_term_by_name function.
 * I did it because I need a special query using vocabulary id
 *
 * @param string $name
 *   term searched
 * @param integer $vid
 *   vocabulary id
 * @return
 *   array of objects for each term found
 */
function agrovocfield_get_term_by_name_and_by_vid($name, $vid) {
  //$db_result = db_query("SELECT t.tid, t.* FROM {term_data} t WHERE
     LOWER(t.name) = LOWER('%s') AND t.vid = '%d' LIMIT 0, 1", trim($name
     ), $vid);
  $args = array(trim($name), $vid);
```

```
  $db_result = db_query_range("SELECT t.tid, t.* FROM {term_data} t WHERE
      LOWER(t.name) = LOWER('%s') AND t.vid = '%d'", $args, 0, 1);
  $result = array();
  while ($term = db_fetch_object($db_result)) {
    $result[] = $term;
  }
  return $result;
}

/**
 * Load a Agrovoc Terms Object
 *
 * @param string $label
 *   Agrovoc term
 * @return
 *   object containing others languages and agrovoc id that belongs to
    this label
 */
function agrovocfield_load_object($label) {
  $term = agrovoc_api_simple_search_by_mode2($label, 'exact', ',', TRUE,
      10);
  $term_labels = agrovoc_api_get_all_labels_by_termcode2($term[0]['id']);
  $languages_supported =  i18n_supported_languages();
  $obj = new stdClass();
  $obj->id = $term[0]['id'];
  foreach ($languages_supported  as $k => $v) {
   // Use substr because FAO Chinese language is ZH and Drupal Chinese
      language is zh-hans,
   // therefore, you  need only 2 first characters.
  $labels[$k] = $term_labels[strtoupper(substr($k, 0, 2))];
  }
  $obj->labels = $labels;
  return $obj;
}

/**
 * Return a label of Agrovoc Terms Object
 *
 * @param object $obj
 *   Agrovoc Terms Object
 * @param string $lang
 *   language, use 2 characters
 * @return
 *   string label in required language
```

```
 */
function agrovocfield_set_label_lang($obj, $lang) {
  return $obj->labels[$lang] ;
}

/**
 * Get translation nid value for a given value of nid
 *
 * @param integer $nid
 *   node id
 * @return
 *   tnid or FALSE
 */
function agrovocfield_get_tnid($nid) {
  $sql = "SELECT n.tnid FROM {node} n WHERE nid = '%d'";
  $tnid = db_result(db_query(db_rewrite_sql($sql, $nid)));
  return $tnid;
}

/**
 * Add terms id to Agrovoc Terms Object
 *
 * @param object $obj
 *   Agrovoc Terms Object
 * @param integer $vid
 *   vocabulary id
 * @return
 *   object modified containing tids stored for each label
 */
function agrovocfield_get_tids_from_labels_object(&$obj, $vid) {
  $terms = $obj->labels;
  // Reseting container.
  unset($obj->labels);
  foreach ($terms as $k => $v) {
    $stored_terms = agrovocfield_get_term_by_name($v);
    foreach ($stored_terms as $stored_term) {
          if (($vid = $stored_term->vid) && ($stored_term->language ==
             $k)) {
            $obj->labels[$stored_term->language] = array(
            'name' => $stored_term->name,
            'tid' => $stored_term->tid,
            );
           }
    }
```

```
  }
  return $obj;
}
Contenido de archivo agrovocfield_autocomplete.info
name = Agrovoc CCK Autocomplete
description =  Defines a autocomplete widget type for Agrovoc CCK field
dependencies[] = agrovocfield
dependencies[] = content
dependencies[] = agrovoc_api
dependencies[] = taxonomy
dependencies[] = i18n
dependencies[] = i18ntaxonomy
package = CCK
core = 6.x
Contenido de archivo agrovocfield_autocomplete.module
<?php
/**
 * @file
 *  Agrovoc CCK field
 *  This module provides an CCK field to fetch and store terms from
    AGROVOC thesaurus
 *  Some portions of code were taken from Content Taxonomy Module and
    adapted to our objectives.
 */

/**
 * Implementation of hook_menu
 */
function agrovocfield_autocomplete_menu() {
  $items['agrovocfield/autocomplete'] = array(
    'title' => 'Autocomplete',
    'page callback' => 'agrovocfield_autocomplete_load',
    'access arguments' => array('access content'),
    'type' => MENU_CALLBACK
  );
  return $items;
}

/**
 * Retrieve a pipe delimited string of autocomplete suggestions
 * Adapted from content_taxonomy_autocomplete module
 *
 * @param String Fieldname
 * @param Integer TID of a parent (optional)
```

```
 * @param BOOLEAN whether a multiple field or not
 * @param STRING typed input
 */
function agrovocfield_autocomplete_load($field_name, $string = '') {
  // The user enters a comma-separated list of tags. We only autocomplete
      the last tag.
  // This regexp allows the following types of user input:
  // this, "somecmpany, llc", "and ""this"" w,o.rks", foo bar
  global $language;
  $content_type_info = _content_type_info();
  $vid = $content_type_info['fields'][$field_name]['vid'];

  // If the menu system has splitted the search text because of slashes,
     glue it back.
  if (func_num_args() > 2) {
    $args = func_get_args();
    $string .= '/'. implode('/', array_slice($args, 2));
  }

  // The user enters a comma-separated list of tags. We only autocomplete
      the last tag.
  $array = drupal_explode_tags($string);

  // Fetch last tag.
  // Get last string and leave $array with a value less.
  $last_string = trim(array_pop($array));
  $prefix = count($array) ? '"'. implode('", "', $array) .'", ' : '';
  $matches = array();

  if ($last_string != '' && strlen($last_string) >= 3) {
       //  use agrovoc webservices
      $agrovoc_result = agrovoc_api_simple_search_by_mode2($last_string,
          'starting', ',', TRUE, 10, $language->language);
      foreach ($agrovoc_result as $term) {
        $n = $term['term'];
        // Commas and quotes in terms are special cases, so encode 'em.
        if (strpos($term['term'], ',') !== FALSE || strpos($term['term'],
            '"') !== FALSE) {
          $n = '"'. str_replace('"', '""', $term['term']) .'"';
        }
        $matches[$prefix . $n] = check_plain($term['term']);
      }
   }
   drupal_json($matches);
```

```
}

/**
 * Implementation of hook_widget_info().
 */
function agrovocfield_autocomplete_widget_info() {
  return array(
  'agrovocfield_autocomplete' => array(
    'label' => t('Agrovoc Autocomplete'),
    'field types' => array('agrovocfield'),
    // Important if you decide to manage multiples values.
    'multiple values' => CONTENT_HANDLE_MODULE,
    'callbacks' => array(
      'default value' => CONTENT_CALLBACK_DEFAULT,
     ),
   ),
  );
  return $items;
}

/**
 * Implementation of hook_widgets_settings().
 */
function agrovocfield_autocomplete_widgets_settings($op, $widget) {
  switch ($op) {
    case 'form':
    case 'save':
  }
}


/**
 * Implementation of hook_elements().
 */
function agrovocfield_autocomplete_elements() {
  return array(
    'agrovocfield_autocomplete' => array(
      '#input' => TRUE,
      '#columns' => array('value'),
      '#delta' => 0,
      '#process' => array('agrovocfield_autocomplete_process'),
      ),
    );
}
```

```
/**
 * Implementation of hook_widget().
 */
function agrovocfield_autocomplete_widget(&$form, &$form_state, $field,
    $items, $delta = NULL) {
  $element = array(
    '#type' => 'agrovocfield_autocomplete',
    '#default_value' => isset($items) ? $items : NULL,
    //set up and show value by default
    '#value_callback' => 'agrovocfield_autocomplete_value',
     '#vid' => $field['vid'],
  );
  return $element;
}

/**
 * show values when edit form is opened
 *
 * returns the taxonomy term name for term ids
 */
function agrovocfield_autocomplete_value($element, $edit = FALSE) {
  $field_key = $element['#columns'][0];
  $values =  agrovocfield_autocomplete_data2form($element['#default_value
      ']);
  return array($field_key => $values);
}

/**
 * Helper function to transpose the stored values to string format for
    each language
 *
 * @param $default_values
 * @return string
 */
function agrovocfield_autocomplete_data2form($default_values) {
    global $language;
    $values = $default_values;
    foreach ($values as $k => $v) {
      // How to show data by default.
      $new_values[] =  agrovocfield_set_label_lang(unserialize($v['value'
          ]), $language->language);
    }
```

```
    $string_values = !is_null($new_values)  ? implode(', ', $new_values) :
        '';
    return $string_values;
}

/**
 * Process an individual element.
 *
 * Build the form element. When creating a form using FAPI #process,
 * note that $element['#value'] is already set.
 *
 */
function agrovocfield_autocomplete_process($element, $edit, $form_state,
    $form) {
  $field_name = $element['#field_name'];
  $field = $form['#field_info'][$field_name];
  $delta = $element['#delta'];
  $field_key = $element['#columns'][0];

  $element['value'] = array(
    '#type' => 'textfield',
    // Used if you have implemented hook_validate.
    '#default_value' => isset($element['#value'][$field_key]) ? $element[
        '#value'][$field_key] : '',
    // Used if you haven't implemented hook_validate.
    // Next commented line is used for debug.
    // '#default_value' => isset($element['#value'][$delta][$field_key])
        ? $element['#value'][$delta][$field_key] : '',
     '#autocomplete_path' => 'agrovocfield/autocomplete/'. $element['#
         field_name'],
    '#title' => $element['#title'],
    '#required' => $element['#required'],
    '#description' => $element['#description'],
    '#field_name' => $element['#field_name'],
    '#type_name' => $element['#type_name'],
    '#delta' => $element['#delta'],
    '#columns' => $element['#columns'],
    '#vid' => $element['#vid'],
    //save old value just in case webservices is disabled
    '#old_value' => $element['#default_value'],
  );

  if (empty($element['value']['#element_validate'])) {
    $element['value']['#element_validate'] = array();
```

```
  }
  array_unshift($element['value']['#element_validate'], '
      agrovocfield_autocomplete_validate');

    if (empty($element['value']['#pre_render'])) {
    $element['value']['#pre_render'] = array();
  }
  array_unshift($element['value']['#pre_render'], '
      agrovocfield_autocomplete_pre_render');

  return $element;
}

/**
 * Validation function for the agrovocfield_autocomplete element
 * Adapted from content_taxonomy_autocomplete module
 * parses input, handles new terms (depending on settings) and sets the
     values as needed for storing the data
 */
function agrovocfield_autocomplete_validate($element, &$form_state) {
  $field_name = $element['#field_name'];
  $field = content_fields($field_name, $element['#type_name']);
  $field_key  = $element['#columns'][0];
  // This segment of code is vital if you want to manage multiples values
      by yourself.
    if ($element['#parents'][count($element['#parents'])-1] == 'value') {
    array_pop($element['#parents']);
    array_pop($element['#array_parents']);
    }
  $value = $element['#value'];
  //check if webservice is working
  if(agrovoc_api_is_available()){
    $extracted_ids = agrovocfield_autocomplete_tags_get_tids($value,
        $field['vid'], $field);
    // Example of how to store values. $values = array(0 => array('value'
        => $element['#value']));
    $values =  agrovocfield_form2data($extracted_ids, $field, $element);
    form_set_value($element, $values, $form_state);
  }
  else {
        //works if webservice is failing
    form_set_value($element, $element['#old_value'], $form_state);
  }
```

```
}

/**
 * Helper function to transpose the values returned by submitting the
    agrovocfield_autocomplete
 * to the format to be stored in the field
 * Adapted from content_taxonomy_autocomplete module
 */
function agrovocfield_form2data($extracted_ids, $field, $element) {
  $existing_tids = is_array($extracted_ids['existing_tids']) ?
     $extracted_ids['existing_tids'] : array();
  $new_tids = array();
  if (is_array($extracted_ids['non_existing_terms'])) {
     $new_tids = agrovocfield_autocomplete_insert_tags($extracted_ids['
        non_existing_terms']);
  }
     if ($input_text !== '') {
     $array_values = array_merge($existing_tids, $new_tids);
     $t = content_transpose_array_rows_cols(array($element['#columns'][0]
        => $array_values));
     return content_transpose_array_rows_cols(array($element['#columns'
        ][0] => $array_values));
     }
    else {
      return array($element['#columns'][0]  => '');
    }
}

/**
 * Show values by default for each language when form node is opened
 *
 */
function agrovocfield_autocomplete_pre_render(&$element) {
  if (isset($_GET['translation']) && isset($_GET['language'])) {
    $translation = $_GET['translation'];
    $language = $_GET['language'];
    $values = explode(', ', $element['#value']);
    foreach ($values as $value) {
      $obj = agrovocfield_load_object($value);
      $translation_value[] = agrovocfield_set_label_lang($obj, $language)
         ;
    }
    $element['#value'] = implode(', ', $translation_value);
  }
```

```
  elseif (!is_null(arg(1)) && is_numeric(arg(1))) {
    $nid = arg(1);
    $translation_nid = agrovocfield_get_tnid($nid);
    if ($translation_nid  != 0 && $nid != $translation_nid && is_numeric(
        $translation_nid )) {
        // Next line is used for debug.
        #$element['#attributes']['disabled'] = TRUE;
        $element['#attributes']['readonly'] = TRUE;
        global $language;
        $translation_node = node_load($translation_nid);
        $translation_field = $translation_node->$element['#field_name'];
        foreach ($translation_field as $k => $v) {
          $obj_value = unserialize($v['value']);
          $new_input[] = agrovocfield_set_label_lang($obj_value,
              $language->language );
        }
        $element['#value'] = implode(', ', $new_input);
     }
  }
  return $element;
}

/**
 * Implementation of hook_theme().
 */
function agrovocfield_autocomplete_theme() {
  return array(
    'agrovocfield_autocomplete' => array(
      'arguments' => array('element' => NULL),
    ),
  );
}


function theme_agrovocfield_autocomplete($element) {
  return $element['#children'];
}

/**
 * Get TIDs for freetagging tags
 * Free tagging vocabularies do not send their tids in the form,
 * so we'll detect them here and process them independently.
 * Taken and adapted from content_taxonomy_autocomplete module
 *
```

```
 * @param $typed_input
 *   A string containing all comma separated tags. As the user typed it.
 * @param integer $vid
 *   vocabulary id
 * @param integer $parent
 *   parent of terms
 * @return
 *   array of terms
 */
function agrovocfield_autocomplete_tags_get_tids($typed_input, $vid,
    $parent = 0) {
  // This regexp allows the following types of user input:
  // this, "somecmpany, llc", "and ""this"" w,o.rks", foo bar
  $typed_terms = agrovocfield_autocomplete_split_tags($typed_input);

  foreach ($typed_terms as $typed_term) {
    // If a user has escaped a term (to demonstrate that it is a group,
    // or includes a comma or quote character), we remove the escape
    // formatting so to save the term into the DB as the user intends.
    $typed_term = trim(str_replace('""', '"', preg_replace('/^"(.*)"$/',
        '\1', $typed_term)));
    if ($typed_term == "") {
        continue; }
    // See if the term exists in the chosen vocabulary
    // and return the tid, otherwise, add a new record.
    // always return one value
    $possibilities = agrovocfield_get_term_by_name_and_by_vid($typed_term
        , $vid);
    // tid match if any.
    $typed_term_tid = NULL;

    foreach ($possibilities as $possibility) {
      $result['existing_tids'][] = serialize(agrovocfield_load_object(
          $possibility->name));
      $typed_term_tid = $possibility->tid;
      }
    if (!$typed_term_tid) {
      $result['non_existing_terms'][] = array(
        'name' => $typed_term,
        'vid' => $vid,
      );
    }
  }
```

```
  return $result;
}

/**
 * Insert new tags
 * Adapted from content_taxonomy_autocomplete module

 * @param array $terms
 *   an array of all <strong>nonexisting</strong> terms.
 * @param integer $parent
 *   parent of terms
 * @return
 *   an array of newly inserted term ids
 */
function agrovocfield_autocomplete_insert_tags($terms, $parent = NULL) {
  module_load_include('inc', 'i18ntaxonomy', 'i18ntaxonomy.admin');
  foreach ($terms as $term) {
    $obj = agrovocfield_load_object($term['name']);
    $trid =  i18ntaxonomy_next_trid();
    $vid = $term['vid'];
    $labels = $obj->labels;
    foreach ( $labels as $lang => $value) {
        $edit = array(
        'name' => $value,
        'vid' => $vid,
        'language' => $lang,
        'trid' => $trid,
        );
    taxonomy_save_term($edit);
    }
    $saved_terms[$trid] =serialize($obj);
  }
  return $saved_terms;
}


/**
 * Helper function to split the tags
 */
function agrovocfield_autocomplete_split_tags($typed_input) {
  $regexp = '%(?:^|,\ *)("(?>[^"]*)(?>""[^"]* )*"|(?: [^",]*))%x';
  preg_match_all($regexp, $typed_input, $matches);
  return $matches[1];
}
```

APPENDIX B

WIDGET PARA EXTRACCIÓN AUTOMÁTICA DE TÉRMINOS DE UN ARCHIVO

Hello, here is some text without a meaning. This text should show, how a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like »Huardest gefburn«. Kjift – Never mind! A blind text like this gives you information about the selected font, how the letters are written and the impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for a special contents, but the length of words should match to the language. Programa en PHP para la implementación del widget de extracción automática de términos de un archivo.

Contenido de archivo agrovocfield_automatic_indexing.info

```
name = Agrovoc CCK Automatic Indexing
description =  Defines a automatic extractor tags from a file for Agrovoc
    field
dependencies[] = agrovocfield
dependencies[] = content
dependencies[] = agrovoc_api
dependencies[] = taxonomy
dependencies[] = i18n
dependencies[] = i18ntaxonomy
package = CCK
core = 6.x
```

Contenido de archivo agrovocfield_automatic_indexing.module

```
<?php


function agrovocfield_automatic_indexing_init(){
  drupal_add_css(drupal_get_path('module', 'agrovocfield') .'/
     agrovocfield.css');
}

/**
 * Implementation of hook_menu().
 */
function agrovocfield_automatic_indexing_menu() {
  $items = array();

  $items['agrovocfield_ai/ahah/%/%/%'] = array(
    'page callback' => 'agrovocfield_ai_js',
```

```
    'page arguments' => array(2, 3, 4),
    'access callback' => TRUE,
    'access arguments' => array('access content'),
    'type' => MENU_CALLBACK,
  );

  return $items;
}

/**
 * Implementation of hook_widget_info().
 */
function agrovocfield_automatic_indexing_widget_info() {
  return array(
  'agrovocfield_automatic_indexing' => array(
    'label' => t('Agrovoc Automatic Indexing'),
    'field types' => array('agrovocfield'),
    // Important if you decide to manage multiples values.
    'multiple values' => CONTENT_HANDLE_MODULE,
    'callbacks' => array(
      'default value' => CONTENT_CALLBACK_DEFAULT,
     ),
   ),
  );
  return $items;
}

/**
 * Implementation of hook_widgets_settings().
 */
function agrovocfield_automatic_indexing_widget_settings($op, $widget) {
  switch ($op) {
    case 'form':
      $form = array();
      $url = isset($widget['url']) ? $widget['url'] : 'http://
          agropedialabs.iitk.ac.in:8080/agroTagger/resources/agrotags';
      $voc_type = isset($widget['voc_type']) ? $widget['voc_type'] :
          array('agrovoc');
      $format_type = isset($widget['format_type']) ? $widget['format_type
          '] : array('text/plain');
      $tags_no = isset($widget['tags_no']) ? $widget['tags_no'] : 5;
      // agrotag or agrovoc
          $form['url'] = array(
            '#title' => t('Webservice URL'),
```

```
            '#type' => 'textfield',
            '#default_value' => $url,
            '#required' => TRUE,
            '#description' => t('Change only if you know new URL provided
                by FAO'),
          );
      $form['voc_type'] = array(
            '#title' => t('Vocabulary Type'),
            '#type' => 'select',
            '#options' => drupal_map_assoc(array('agrotag', 'agrovoc')),
            '#default_value' => $voc_type,
            '#required' => TRUE,
          );
          // xml/rdf, application/JSON, text/plain
      $form['format_type'] = array(
        '#title' => t('Output Type'),
        '#type' => 'select',
        '#options' => drupal_map_assoc(array('text/plain')),
            '#default_value' => $format_type,
        '#required' => TRUE,
      );
      // 10, 15, 20 ...
      $form['tags_no'] = array(
        '#title' => t('No. of Tags'),
        '#type' => 'select',
        '#options' => drupal_map_assoc(array(5, 10, 15)),
        '#default_value' => $tags_no,
        '#required' => TRUE,
      );
      return $form;
    case 'save':
      return array('url', 'voc_type', 'format_type', 'tags_no');
  }
}

/**
 * Implementation of hook_elements().
 */
function agrovocfield_automatic_indexing_elements() {
  return array(
    'agrovocfield_automatic_indexing' => array(
      '#input' => TRUE,
      '#columns' => array('value'),
      '#delta' => 0,
```

```
      '#process' => array('agrovocfield_automatic_indexing_process'),
      ),
    );
}

/**
 * show values when edit form is opened
 *
 * returns the taxonomy term name for term ids
 */
function agrovocfield_automatic_indexing_value($element, $edit = FALSE) {
  $field_key = $element['#columns'][0];
  $values =  agrovocfield_autocomplete_data2form($element['#default_value
      ']);
  return array($field_key => $values);
}

/**
 * Process an individual element.
 *
 * Build the form element. When creating a form using FAPI #process,
 * note that $element['#value'] is already set.
 *
 */
function agrovocfield_automatic_indexing_process($element, $edit,
   $form_state, $form) {
  $field_name = $element['#field_name'];
  $field = $form['#field_info'][$field_name];
  $delta = $element['#delta'];
  $field_key = $element['#columns'][0];

  $element['value'] = array(
    '#type' => 'textfield',
    // Used if you have implemented hook_validate.
    '#default_value' => isset($element['#value'][$field_key]) ? $element[
       '#value'][$field_key] : '',
    // Used if you haven't implemented hook_validate.
    // Next commented line is used for debug.
    // '#default_value' => isset($element['#value'][$delta][$field_key])
       ? $element['#value'][$delta][$field_key] : '',
    '#title' => $element['#title'],
    '#required' => $element['#required'],
    '#description' => $element['#description'],
    '#field_name' => $element['#field_name'],
```

```
    '#type_name' => $element['#type_name'],
    '#delta' => $element['#delta'],
    '#columns' => $element['#columns'],
    '#vid' => $element['#vid'],
    //save old value just in case webservices is disabled
    '#old_value' => $element['#default_value'],
    '#prefix' => t('<div id="@element-agrovocfield-ai-ahah-wrapper">',
       array("@element" => $element['#id'])),
    '#suffix' => '</div>',
  );

  $element['agrovocfield_ai_upload'] = array(
    '#type' => 'file',
    '#description' => t('Select file to extract Agrovoc tags
       automatically'),
    '#prefix' => t('<div class="agrovocfield-ai-container">'),
  );

  $element['agrovocfield_ai_submit'] = array(
    '#type' => 'submit',
    '#description' => t('Retrieve Agrovoc tags'),
    '#value' => t('Retrieve tags'),
    //'#submit' => array('node_form_submit_build_node'),
    '#ahah' => array( // with JavaScript
       'path' => 'agrovocfield_ai/ahah/'.   $element['#type_name'] .'/'.
          $element['#field_name'] .'/'. $element['#delta'],
       'wrapper' => $element['#id'] .'-agrovocfield-ai-ahah-wrapper',
       'method' => 'replace',
       'effect' => 'fade',
    ),
    '#field_name' => $element['#field_name'],
    '#delta' => $element['#delta'],
    '#type_name' => $element['#type_name'],
    '#weight' => 100,
    '#post' => $element['#post'],
    '#suffix' => '</div>',
  );


  if (empty($element['value']['#element_validate'])) {
    $element['value']['#element_validate'] = array();
  }
  array_unshift($element['value']['#element_validate'], '
     agrovocfield_autocomplete_validate');
```

```
  return $element;
}

/**
 * Implementation of hook_widget().
 */
function agrovocfield_automatic_indexing_widget(&$form, &$form_state,
    $field, $items, $delta = NULL) {
  $element = array(
    '#type' => 'agrovocfield_automatic_indexing',
    '#default_value' => isset($items) ? $items : NULL,
    //set up and show value by default
    '#value_callback' => 'agrovocfield_automatic_indexing_value',
     '#vid' => $field['vid'],
  );
  return $element;
}

/**
 * Implementation of hook_theme().
 */
function agrovocfield_automatic_indexing_theme() {
  return array(
    'agrovocfield_automatic_indexing' => array(
      'arguments' => array('element' => NULL),
    ),
  );
}

function theme_agrovocfield_automatic_indexing($element) {
  return $element['#children'];
}

/**
 * Menu callback; Shared AHAH callback for uploads and deletions.
 *
 * This rebuilds the form element for a particular field item. As long as
     the
 * form processing is properly encapsulated in the widget element the
    form
 * should rebuild correctly using FAPI without the need for additional
    callbacks
 * or processing.
```

```
 */
function agrovocfield_ai_js($type_name, $field_name, $delta) {
 $field = content_fields($field_name, $type_name);
 $data = array();
 if(isset($_FILES['files']['name'][$field_name]) && is_uploaded_file(
    $_FILES['files']['tmp_name'][$field_name])){
        $filename = $_FILES['files']['name'][$field_name];
        $data['url'] = $field['widget']['url'];
        //store temporal file
        $data['file'] = file_save_upload($field_name, '',
           file_directory_path());
        // agrotag or agrovoc
        $data['voc_type'] = $field['widget']['voc_type'];
        // xml/rdf, application/JSON, text/plain
    $data['format_type'] = $field['widget']['format_type'];
    // 10, 15, 20 ...
    $data['tags_no'] = $field['widget']['tags_no'];
 }
 $tags = agrovocfield_automatic_indexing_curl($data);
  //deleting temporal file
 $success = file_delete($data['file']->filepath);
  //If the file was successfully deleted, update the database
  if ($success){
    db_query('DELETE FROM {files} WHERE filepath = "%s"', $data['file']->
       filepath);
  }
 // Immediately disable devel shutdown functions so that it doesn't botch
     our
  // JSON output.
 $GLOBALS['devel_shutdown'] = FALSE;

  if (empty($field) || empty($_POST['form_build_id'])) {
    // Invalid request.
    drupal_set_message(t('An unrecoverable error occurred. The uploaded
       file likely exceeded the maximum file size (@size) that this
       server supports.', array('@size' => format_size(
       file_upload_max_size()))), 'error');
    print drupal_to_js(array('data' => theme('status_messages')));
    exit;
  }

  // Build the new form.
  $form_state = array('submitted' => FALSE);
  $form_build_id = $_POST['form_build_id'];
```

```
  $form = form_get_cache($form_build_id, $form_state);

  if (!$form) {
    // Invalid form_build_id.
    drupal_set_message(t('An unrecoverable error occurred. This form was
        missing from the server cache. Try reloading the page and
        submitting again.'), 'error');
    print drupal_to_js(array('data' => theme('status_messages')));
    exit;
  }

  // Build the form. This calls the file field's #value_callback function
      and
  // saves the uploaded file. Since this form is already marked as cached
  // (the #cache property is TRUE), the cache is updated automatically
      and we
  // don't need to call form_set_cache().
  $args = $form['#parameters'];
  $form_id = array_shift($args);
  $form['#post'] = $_POST;
  $form = form_builder($form_id, $form, $form_state);

  if (!isset($form_element)) {
    $form_element = $form[$field_name]['value'];
  }

  $form_element['#value'] = $tags;

  if (isset($form_element['_weight'])) {
    unset($form_element['_weight']);
  }

  $output = drupal_render($form_element);

  // For some reason, file uploads don't like drupal_json() with its
      manual
  // setting of the text/javascript HTTP header. So use this one instead.
  print drupal_to_js(array('status' => TRUE, 'data' => $output));
  exit;
}
```

```
function agrovocfield_automatic_indexing_form_alter(&$form, $form_state,
    $form_id) {
  // Field configuration (for default images).
  if ($form_id == 'content_field_edit_form' && isset($form['#field']) &&
      $form['#field']['type'] == 'agrovocfield') {
    $form['#attributes']['enctype'] = 'multipart/form-data';
  }

  // Node forms.
  if (preg_match('/_node_form$/', $form_id)) {
    $form['#attributes']['enctype'] = 'multipart/form-data';
  }
}



/**
 * Enter description here ...
 * @param unknown_type $file
 * @return mixed
 */
function agrovocfield_automatic_indexing_curl($values){
  $url = $values['url'];
  $data = array();
  $filepath = realpath('./') . '/' . $values['file']->filepath;
  $data['file_url'] = "@$filepath";
  // agrotag or agrovoc
  $data['voctype'] = $values['voc_type'];
  // xml/rdf, application/JSON, text/plain
  $data['outputtype'] = $values['format_type'];
  // 10, 15, 20 ...
  $data['tags_no'] = $values['tags_no'];
        //start curl process
  $ch = curl_init();
  $result=curl_setopt($ch, CURLOPT_URL, $url);
  $result=curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
  $result=curl_setopt($ch, CURLOPT_POST, 1 );
  $result=curl_setopt($ch, CURLOPT_POSTFIELDS, $data );
  $http_result = curl_exec($ch);
  return $http_result;
}
Contenido de archivo agrovocfield.css
.agrovocfield-ai-container {
display:inline-block;
}
```

```
.agrovocfield-ai-container .form-item,
.agrovocfield-ai-container .form-submit {
display:inline-block;
margin: 0px;
float:left;
}
```