UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

DESARROLLO DE UNA HERRAMIENTA DE ANÁLISIS DE METADATOS DE ACCESIBILIDAD

Daniel Ortega Moreno

Septiembre 2019

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENIERÍA INFORMÁTICA

Trabajo Fin de Carrera

TÍTULO DEL TFG

Autor: **Daniel Ortega Moreno**

Director: **Salvador Otón Tortosa**

Tribunal:

Presidente: _____

Vocal 1º: _____

Vocal 2º: _____

Calificación: _____

Alcalá de Henares a de de 2019

ÍNDICE RESUMIDO

ÍNDICE DETALLADO

INTRODUCCIÓN

El acceso a internet es actualmente una necesidad para el desarrollo de la vida en gran parte del mundo, por lo que es necesario que todas las personas sean capaces de aprovechar las ventajas y oportunidades que proporciona este medio.

Sin embargo, un gran porcentaje de las páginas web que existen no proporcionan los medios adecuados para que un sector de la población pueda interactuar de forma sencilla e intuitiva con ellas.

La "accesibilidad" web es una práctica que tanto los creadores de estas páginas web como los desarrolladores de navegadores y dispositivos deben incluir en sus creaciones para eliminar las barreras que puedan existir para cualquier persona a la hora de utilizarlas.

Es importante poder realizar un seguimiento del uso de esta práctica ya que en algunos casos puede ser un requisito legal pero actualmente no existe ninguna herramienta que permita realizar un estudio sobre una gran cantidad de páginas de forma simple.

INTRODUCTION

Internet Access is nowadays a necessity for life development in most of the world, so it is necessary that everyone can make use of the advantages and opportunities that this platform has to offer.

However, a large percentage of the available websites do not provide the proper means for a portion of the population to be able to interact in a simple and intuitive way with them.

Web "accessibility" is a practice that websites and browsers developers should include to eliminate the barriers that some people may face when using these products.

It is important to be able to track the use of this practice that may be a legal requisite in some cases but currently a tool that allows to study a large number of webpages in a simple way doesn´t exist.

PALABRAS CLAVE

Accesibilidad, Metadatos, Common Crawl, Web Data Commons, Microdatos, RDFa, JSON-LD

KEYWORDS

Accessibility, Metadata, Common Crawl, Web Data Commons, Microdata, RDFa, JSON-LD

OBJETIVOS DEL PROYECTO

Con el objetivo de poder analizar grandes cantidades de información sobre páginas web para estudiar el uso de metadatos de accesibilidad a través de internet se pretende desarrollar una herramienta que permita automatizar este proceso.

Para el desarrollo de esta herramienta se han establecido los siguientes pasos:

- Estudio y comprensión de los esquemas proporcionados por schema.org en relación con la accesibilidad, sus posibles valores, significado e inclusión en el lenguaje HTML.
- Estudio de la estructura de la información obtenida gracias al rastreo de páginas web que ofrece la organización Common Crawl para su uso público con el fin de facilitar el acceso de la herramienta a esta información.
- Estudio de otros datasets de los que se pueda obtener la información que se busca en este proyecto.
- Estudio del funcionamiento del framework de procesamiento del dataset de Common Crawl que ofrece la organización Web Data Commons con el fin de modificarlo para extraer los datos deseados.
- Modificación del framework mencionado para permitir el procesamiento de datos tanto en la nube como en una máquina local.
- Desarrollo de extractores de datos para los datasets que se hayan considerado aptos para su análisis.
- Creación de una interfaz simple que permita visualizar ciertos datos básicos sobre la información obtenida.

ESTADO DEL ARTE

1 Common Crawl

El conjunto de datos elegido para el análisis de estos metadatos es el que proporciona de forma gratuita Common Crawl.

La Fundación Common Crawl es una organización sin ánimo de lucro con el objetivo de democratizar el acceso a la información de internet mediante la producción y el mantenimiento de un repositorio abierto de los contenidos de la web que sean universalmente accesibles y analizables.

Estos datos forman petabytes de información recogida a lo largo de 7 años y se amplía mensualmente desde marzo de 2014.

Forman parte del programa de Amazon *AWS Public Dataset Program* por lo que todos los archivos son accesibles mediante los protocolos HTTP y S3.

Debido a que los datos se dividen en un gran número de archivos, las rutas a estos son accesibles mediante documentos que contienen listas de rutas a estos. Estos archivos de *paths* son accesibles mediante los protocolos ya mencionados mediante las rutas s3://commoncrawl/ para S3 y https://commoncrawl.s3.amazonaws.com/ para HTTP seguidas de una cadena con formato CC-MAIN-YYYY-DD/[segment|warc|wat|wet].paths.gz donde YYYY representa el año, DD la semana del año en la que se almacenaron los datos (generalmente la última de cada mes), y segment, warc, wat y wet representan el conjunto de archivos a los que dirigen estas rutas, los cuales se explicarán más adelante.

Formato de datos

La información contenida en Common Crawl para cada web se almacena en 3 formatos de datos distintos, todos basados en el estándar WARC (Web ARChive), que permite una mayor eficiencia a la hora de almacenar y procesar los archivos consistentes de miles de millones de páginas web ya que pueden alcanzar un tamaño de cientos de terabytes.

Hasta la fecha Common Crawl sigue el estándar 1.0 de este formato que se puede encontrar en: http://bibnum.bnf.fr/WARC/WARC_ISO_28500_version1_latestdraft.pdf.

WARC

El formato WARC contiene toda la información en bruto del rastreo, proporcionando un acceso directo al proceso de este. No solo almacena la respuesta HTTP de las páginas con las que contacta, sino también datos sobre cómo se ha solicitado esa información, así como metadatos del proceso.

Estos tres tipos de información se pueden distinguir mediante el campo WARC-Type de cada registro ya que a cada una se le asigna el valor response, request y metadata respectivamente.

La información almacenada para las respuestas HTTP no solo incluye la información devuelta que se obtendría al descargar el archivo, sino también las cabeceras de este protocolo, que pueden ser útiles para profundizar en el análisis de lo anterior.

WARC/1.0

WARC-Type: response

WARC-Date: 2014-08-02T09:52:13Z

WARC-Record-ID: <urn:uuid:ffbfb0c0-6456-42b0-af03-3867be6fc09f>

Content-Length: 43428

Content-Type: application/http; msgtype=response

WARC-Warcinfo-ID: <urn:uuid:3169ca8e-39a6-42e9-a4e3-9f001f067bdf>

WARC-Concurrent-To: <urn:uuid:d99f2a24-158a-4c77-bb0a-3cccd40aad56>

WARC-IP-Address: 212.58.244.61

WARC-Target-URI: http://news.bbc.co.uk/2/hi/africa/3414345.stm

WARC-Payload-Digest: sha1:M63W6MNGFDWXDSLTHF7GWUPCJUH4JK3J

WARC-Block-Digest: sha1:YHKQUSBOS4CLYFEKQDVGJ457OAPD6IJO

WARC-Truncated: length

HTTP/1.1 200 OK

Server: Apache

Vary: X-CDN

Cache-Control: max-age=0

Content-Type: text/html

Date: Sat, 02 Aug 2014 09:52:13 GMT

Expires: Sat, 02 Aug 2014 09:52:13 GMT

Connection: close

Set-Cookie: BBC-UID=15730d9c1b741c0d3942e2aca1317fbf39e57b90be68a329d375ba9d5a8964080CCBot%2f2%2e0%20%28http%3a%2f%2fcommoncrawl%2eorg%2ffaq%2f%29; expires=Sun, 02-Aug-15 09:52:13 GMT; path=/; domain=bbc.co.uk;

Set-Cookie: BBC-UID=15730d9c1b741c0d3942e2aca1317fbf39e57b90be68a329d375ba9d5a8964080CCBot%2f2%2e0%20%28http%3a%2f%2fcommoncrawl%2eorg%2ffaq%2f%29; expires=Sun, 02-Aug-15 09:52:13 GMT; path=/; domain=bbc.co.uk;

<!doctype html public "-//W3C//DTD HTML 4.0 Transitional//EN" "http://www.w3.org/TR/REC-html40/loose.dtd">

<html>

```
<head>
<title>
        BBC NEWS | Africa | Namibia braces for Nujoma exit
</title>
<meta name="keywords" content="BBC, News, BBC News, news online, world, uk, international, foreign, british, online, service" />
        …
<body bgcolor="#ffffff" text="#000000">


<a name="top" id="top"></a>
<!-- START TOOLBAR -->
        <div id="ift-toolbar" class="ift-news">
        …
</body>
```

WAT

Lor archivos WAT almacenan metadatos importantes sobre los registros almacenados en formato WARC. Estos metadatos son extraídos de cada uno de los tres tipos de registros (metadata, request y response). Si la información es HTML, la extracción incluye las cabeceras HTTP y los enlaces (incluyendo el tipo de enlace) contenidos en la página.

La información se almacena en formato JSON. Para mantener el tamaño de los archivos se eliminan de este formato todos los espacios en blanco innecesarios, creando una cadena relativamente ilegible para un humano.

El esquema del JSON del formato WAT es:

```
Envelope
  WARC-Header-Metadata
    WARC-Target-URI [string]
    WARC-Type [string]
    WARC-Date [datetime string]
    ...
  Payload-Metadata
    HTTP-Response-Metadata
      Headers
        Content-Language
        Content-Encoding
        ...
      HTML-Metadata
        Head
          Title [string]
          Link [list]
          Metas [list]
        Links [list]
      Headers-Length [int]
      Entity-Length [int]
      ...
    ...
  ...
Container
  Gzip-Metadata [object]
  Compressed [boolean]
  Offset [int]
```

Ejemplo completo de WAT extraído de http://news.bbc.co.uk/2/hi/africa/3414345.stm:

```
{
  "Envelope" : {
    "WARC-Header-Length" : "578",
    "Block-Digest" : "sha1:YHKQUSBOS4CLYFEKQDVGJ457OAPD6IJO",
    "Format" : "WARC",
    "Actual-Content-Length" : "43428",
    "WARC-Header-Metadata" : {
      "WARC-Record-ID" : "<urn:uuid:ffbfb0c0-6456-42b0-af03-3867be6fc09f>",
      "WARC-Warcinfo-ID" : "<urn:uuid:3169ca8e-39a6-42e9-a4e3-9f001f067bdf>",
      "Content-Length" : "43428",
      "WARC-Date" : "2014-08-02T09:52:13Z",
      "Content-Type" : "application/http; msgtype=response",
      "WARC-Target-URI" : "http://news.bbc.co.uk/2/hi/africa/3414345.stm",
      "WARC-IP-Address" : "212.58.244.61",
      "WARC-Block-Digest" : "sha1:YHKQUSBOS4CLYFEKQDVGJ457OAPD6IJO",
      "WARC-Payload-Digest" : "sha1:M63W6MNGFDWXDSLTHF7GWUPCJUH4JK3J",
      "WARC-Truncated" : "length",
      "WARC-Concurrent-To" : "<urn:uuid:d99f2a24-158a-4c77-bb0a-3cccd40aad56>",
      "WARC-Type" : "response"
    },
    "Payload-Metadata" : {
      "Actual-Content-Type" : "application/http; msgtype=response",
      "Trailing-Slop-Length" : "4",
      "HTTP-Response-Metadata" : {
        "Entity-Length" : "42809",
        "Headers-Length" : "619",
        "Entity-Trailing-Slop-Bytes" : "0",
        "Entity-Digest" : "sha1:M63W6MNGFDWXDSLTHF7GWUPCJUH4JK3J",
        "Response-Message" : {
          "Version" : "HTTP/1.1",
          "Reason" : "OK",
          "Status" : "200"
        },
        "HTML-Metadata" : {
```

```
"Links" : [
  {
    "href" : "/css/screen/shared/styles.css",
    "path" : "STYLE/#text"
  },
  {
    "href" : "/css/screen/shared/toolbar_ifs.css",
    "path" : "STYLE/#text"
  },
        …
],
"Head" : {
  "Title" : " BBC NEWS | Africa | Namibia braces for Nujoma exit ",
  "Link" : [
    {
      "rel" : "stylesheet",
      "type" : "text/css",
      "url" : "/css/screen/nol/story.css?v1",
      "path" : "LINK@/href"
    }
  ],
  "Scripts" : [
    {
      "type" : "text/javascript",
      "url" : "http://news.bbcimg.co.uk/js/newsi/latest/newsi.js?9",
      "path" : "SCRIPT@/src"
    },
    {
      "type" : "text/javascript",
      "url" : "http://www.bbc.co.uk/glow/gloader.js",
      "path" : "SCRIPT@/src"
    },
  …
  ],
  "Metas" : [
```

```
            {
              "name" : "keywords",
              "content" : "BBC, News, BBC News, news online, world, uk, international, foreign, british, online, service"
            },
            {
              "content" : "2004/01/22 00:48:49",
              "name" : "OriginalPublicationDate"
            },
          …
           ]
         }
       },
       "Headers" : {
         "Date" : "Sat, 02 Aug 2014 09:52:13 GMT",
         "Cache-Control" : "max-age=0",
         "Connection" : "close",
         "Expires" : "Sat, 02 Aug 2014 09:52:13 GMT",
         "Content-Type" : "text/html",
         "Server" : "Apache",
         "Vary" : "X-CDN",
        …
       }
     }
  }
},
"Container" : {
  "Compressed" : true,
  "Offset" : "213390650",
  "Filename" : "CC-MAIN-20140728011800-00009-ip-10-146-231-18.ec2.internal.warc.gz",
  "Gzip-Metadata" : {
    "Footer-Length" : "8",
    "Header-Length" : "10",
    "Inflated-Length" : "44010",
    "Deflate-Length" : "11830",
    "Inflated-CRC" : "1006775323"
```

```
    }
  }
}
```

WET

El formato WET recoge únicamente texto plano para aquellas tareas que sólo requieran información textual. En este caso el modo en el que se almacena la información es muy sencillo, tras varios detalles de metadatos, incluyendo la URL de la página y la longitud del texto, se encuentra el texto en sí justo después.

```
WARC/1.0
WARC-Type: conversion
WARC-Target-URI: http://news.bbc.co.uk/2/hi/africa/3414345.stm
WARC-Date: 2014-08-02T09:52:13Z
WARC-Record-ID:
WARC-Refers-To:
WARC-Block-Digest: sha1:JROHLCS5SKMBR6XY46WXREW7RXM64EJC
Content-Type: text/plain
Content-Length: 6724

BBC NEWS | Africa | Namibia braces for Nujoma exit
[an error occurred while processing this directive]
Low graphics|Accessibility help
One-Minute World News
...
President Sam Nujoma works in very pleasant surroundings in the small but beautiful old State House...
```

2 Schema.org

El formato elegido para la inserción de los microdatos de accesibilidad ha sido el que propone Schema.org.

Schema.org fue fundada por Google, Microsoft, Yahoo y Yandex, y hoy en día tiene detrás a toda una comunidad colaborando para crear, mantener y promover esquemas para datos estructurados en Internet, entre otros, páginas web como es nuestro caso. Los vocabularios son desarrollados a través de un proceso de comunidad abierta, usando la lista de e-mail public-schemaorg@w3.org y a través de GitHub.

Hoy en día, muchos sitios y empresas como Google, Microsoft o Pinterest entre otros, utilizan este formato para marcar sus páginas web y correos electrónicos. Además, estas propiedades se utilizan por sus aplicaciones para obtener una mejor experiencia.

La idea fundamental que se promueve es que un vocabulario compartido hace más fácil a los desarrolladores decidir sobre un esquema y obtener el máximo beneficio de sus esfuerzos.

Se proporciona una colección compartida de esquemas que cualquier usuario puede consultar a través de su página web.

Esquemas de accesibilidad

De entre todos los esquemas, nos interesan aquellos que están relacionados con la accesibilidad para este proyecto.

Dentro de CreativeWork se encuentran las propiedades necesarias para identificar un recurso en materia de accesibilidad ya que dentro de esta categoría se encuentran otras como Book, WebPage, Article…

A continuación, se detallan las propiedades relacionadas con accesibilidad en Schema.org para CreativeWork:

AccessMode

Descripción	***Definición***
Nombre del atributo	***accessMode***
Tipo de dato	Text
Espacio valor	auditory chartOnVisual chemOnVisual

	colorDependent diagramOnVisual mathOnVisual musicOnVisual tactile textOnVisual textual visual
Descripción	El sistema perceptivo sensorial humano o facultad cognitiva a través del cual una persona puede procesar o percibir información.

AccessModeSufficient

Descripción	***Definición***
Nombre del atributo	***accessModeSufficient***
Tipo de dato	Text
Espacio valor	auditory tactile textual visual
Descripción	Una lista de accessModes únicos o combinados que son suficientes para entender todo el contenido intelectual de un recurso.

AccessibilitySummary

Descripción	*Definición*
Nombre del atributo	***accessibilitySummary***
Tipo de dato	Text
Espacio valor	*Resumen legible de las características y deficiencias de accesibilidad.*
Descripción	Un resumen legible por humanos de características o deficiencias específicas de accesibilidad, consistente con los otros metadatos de accesibilidad, pero que expresa sutilezas tales como "Están presentes descripciones cortas, pero se necesitarán descripciones largas para usuarios no visuales" o "Están presentes descripciones cortas y no se necesitan descripciones largas."

AccessibilityFeature

Descripción	*Definición*
Nombre del atributo	***accessibilityFeature***
Tipo de dato	Text
Espacio valor	alternativeText annotations audioDescription bookmarks braille captions ChemML describedMath

	displayTransformability highContrastAudio highContrastDisplay index largePrint latex longDescription MathML none printPageNumbers readingOrder rubyAnnotations signLanguage structuralNavigation synchronizedAudioText tableOfContents taggedPDF tactileGraphic tactileObject timingControl transcript ttsMarkup unlocked
Descripción	Las características del contenido del recurso, como el contenido multimedia accesible, las mejoras soportadas para la accesibilidad y las alternativas.

AccessibilityHazard

Descripción	*Definición*
Nombre del atributo	***accessibilityHazard***
Tipo de dato	Text
Espacio valor	flashing noFlashingHazard motionSimulation noMotionSimulationHazard sound noSoundHazard unknown
Descripción	Una característica del recurso descrito que es fisiológicamente peligrosa para algunos usuarios. Relacionado con la guía 2.3 de WCAG 2.0. Las tres propiedades negativas deben establecerse si no se conoce ninguno de los peligros. Si el contenido tiene peligro (s), incluya aserciones positivas para los peligros que tiene y afirmaciones negativas para los demás. Si la propiedad no está configurada en positivo o negativo o se define específicamente como desconocida, el estado de los peligros no se conoce.

AccessibilityControl

Descripción	*Definición*
Nombre del atributo	***accessibilityControl***
Tipo de dato	Text

Espacio valor	fullKeyboardControl fullMouseControl fullSwitchControl fullTouchControl fullVideoControl fullVoiceControl
Descripción	Identifica uno o más métodos de entrada que permiten el acceso a toda la funcionalidad de la aplicación.

AccessibilityAPI

Descripción	***Definición***
Nombre del atributo	***accessibilityAPI***
Tipo de dato	Text
Espacio valor	AndroidAccessibility ARIA ATK AT-SPI BlackberryAccessibility iAccessible2 iOSAccessibility JavaAccessibility MacOSXAccessibility MSAA UIAutomation
Descripción	Indica que el recurso es compatible con la API de accesibilidad referenciada.

Formato de datos

A parte de definir esquemas y propiedades también se debe definir una forma común de integrar esta información junto al contenido.

Schema.org propone tres formatos diferentes para incluir sus esquemas en páginas web.

Microdata

Web Hypertext Application Technology Working Group (WHATWG) es una comunidad fundada por miembros de Apple, Mozilla Foundation y opera Software a la que más tarde se uniría Google, orientada al desarrollo de HTML y tecnologías relacionadas.

Microdata es una especificación de WHATWG usada para el anidamiento de metadatos en el contenido de páginas web existentes.

Microdata consiste de un grupo de pares clave-valor. Los grupos se llaman ítems (objetos) y cada par clave-valor es una propiedad.

Los posibles atributos que definen estos objetos son:

- Itemscope: crea el objeto e indica que los descendientes de este elemento contienen información sobre este.
- Itemtype: url al vocabulario que define el contexto del objeto y sus propiedades.
- Itemid: identificador único del objeto.
- Itemprop: indica que el elemento HTML en el que está definido contiene el valor de la propiedad del objeto especificada. El contexto del nombre y valor de la propiedad están descritos en el vocabulario del objeto. Los valores generalmente consisten de cadenas, pero también pueden ser URLs usando el elemento *a* y su atributo *href*, el elemento *img* y su atributo *src*…
- Itemref: las propiedades que no son descendientes del elemento con el atributo itemscope pueden ser asociados con el objeto usando este atributo.
- Datetime: indica fecha o duración.

A continuación se muestra un ejemplo de microdata con metadatos de accesibilidad:

```
<div itemscope="" itemtype="http://schema.org/Book">
  <meta itemprop="bookFormat" content="EBook/DAISY3"/>
  <meta itemprop="accessibilityFeature" content="largePrint/CSSEnabled"/>
```

```
  <meta itemprop="accessibilityFeature" content="highContrast/CSSEnabled"/>
  <meta itemprop="accessibilityFeature" content="resizeText/CSSEnabled"/>
  <meta itemprop="accessibilityFeature" content="displayTransformability"/>
  <meta itemprop="accessibilityFeature" content="longDescription"/>
  <meta itemprop="accessibilityFeature" content="alternativeText"/>
  <meta itemprop="accessibilityControl" content="fullKeyboardControl"/>
  <meta itemprop="accessibilityControl" content="fullMouseControl"/>
  <meta itemprop="accessibilityHazard" content="noFlashingHazard"/>
  <meta itemprop="accessibilityHazard" content="noMotionSimulationHazard"/>
  <meta itemprop="accessibilityHazard" content="noSoundHazard"/>
  <meta itemprop="accessibilityAPI" content="ARIA"/>
  <dl>
    <dt>Name:</dt>
    <dd itemprop="name">Holt Physical Science</dd>
    <dt>Brief Synopsis:</dt>
    <dd itemprop="description">NIMAC-sourced textbook</dd>
  </dl>
</div>
```

RDFa

Resource Description Framework in Attributes es una especificación del World Wide Web Consortium (W3C) que añade un conjunto de extensiones a HTML, XHTML y varios formatos basados en XML para incluir metadatos en documentos web.

Los atributos utilizados en este formato son:

- Vocab: define una URL al vocabulario o vocabularios que van a ser utilizados dentro del fragmento HTML donde se encuentra este atributo.
- Typeof: declara un nuevo objeto del tipo indicado en el valor de este atributo.
- Property: especifica una propiedad de un elemento.
- Content: atributo que cuando acompaña a property reemplaza al contenido del elemento y define el valor de la propiedad anteriormente especificada.

- Datatype: atributo que cuando acompaña a property señala el tipo de datos a usar en la propiedad especificada.
- Rel y rev: indican relación con orto recurso.
- Src, href y resource especifican la ubicación del recurso definido.

A continuación, se muestra un ejemplo de metadatos de accesibilidad indicados mediante RDFa:

```
<div vocab="http://schema.org/" typeof="Book">
  <meta property="bookFormat" content="EBook/DAISY3"/>
  <meta property="accessibilityFeature" content="largePrint/CSSEnabled"/>
  <meta property="accessibilityFeature" content="highContrast/CSSEnabled"/>
  <meta property="accessibilityFeature" content="resizeText/CSSEnabled"/>
  <meta property="accessibilityFeature" content="displayTransformability"/>
  <meta property="accessibilityFeature" content="longDescription"/>
  <meta property="accessibilityFeature" content="alternativeText"/>
  <meta property="accessibilityControl" content="fullKeyboardControl"/>
  <meta property="accessibilityControl" content="fullMouseControl"/>
  <meta property="accessibilityHazard" content="noFlashingHazard"/>
  <meta property="accessibilityHazard" content="noMotionSimulationHazard"/>
  <meta property="accessibilityHazard" content="noSoundHazard"/>
  <meta property="accessibilityAPI" content="ARIA"/>
  <dl>
    <dt>Name:</dt>
    <dd property="name">Holt Physical Science</dd>
    <dt>Brief Synopsis:</dt>
    <dd property="description">NIMAC-sourced textbook</dd>
 </dl>
</div>
```

JSON-LD

La información enlazada, linked data o LD es un tipo de datos estructurado definido en 2006 por el World Wide Web Consortium (W3C) que se entrelaza con otra información para proporcionarla de sentido semántico.

JavaScript Object Notation for Linked Data (JSON-LD) es un método de codificar linked data utilizando JSON. Está diseñado alrededor del concepto de un contexto que permite relacionar atributos de JSON a un modelo de Resource Description Framework (RDF). Con el fin de relacionar la sintaxis de JSON-LD con RDF, JSON-LD permite que los valores sean forzados a un tipo especificado o etiquetados mediante un lenguaje.

Los atributos principales que forman parte de este lenguaje son:

@context: define los nombres que van a ser utilizados a lo largo del documento. Estos nombres se denominan términos y permiten a los desarrolladores expresar identificadores específicos de forma compacta.

@type: especifica el tipo de datos de un nodo.

@id: permite identificar de forma unívoca las cosas que se describen en el documento.

@value: especifica la información asociada con una propiedad concreta.

A continuación, se muestra un ejemplo de metadatos de accesibilidad representados mediante JSON-LD:

```
<script type="application/ld+json">
{
 "@context": "http://schema.org",
 "@type": "Book",
 "accessibilityAPI": "ARIA",
 "accessibilityControl": [
  "fullKeyboardControl",
  "fullMouseControl"
 ],
 "accessibilityFeature": [
  "largePrint/CSSEnabled",
  "highContrast/CSSEnabled",
  "resizeText/CSSEnabled",
  "displayTransformability",
  "longDescription",
  "alternativeText"
```

```
  ],
  "accessibilityHazard": [
    "noFlashingHazard",
    "noMotionSimulationHazard",
    "noSoundHazard"
  ],
  "bookFormat": "EBook/DAISY3",
  "description": "NIMAC-sourced textbook",
  "name": "Holt Physical Science",
}
</script>
```

3 Web Data Commons

Web Data Commons es un proyecto dedicado a la extracción de información estructurada de los datasets de Common Crawl y proporciona los datos extraídos al público con el fin de apoyar a investigadores y compañías a explotar la gran cantidad de información disponible en la web. Además, proporciona un framework basado en Java y Amazon Web Services (AWS) con el que se facilita la extracción de información de forma personalizada.

Dataset de metadatos

En este dataset se almacena información extraída de Common Crawl que pertenece a distintas especificaciones de integración de semántica en HTML. Estas especificaciones son: RDFa, Microdata, hCard, hCalendar, Geo, hListing, hResume, hReview, hRecipe, Species, XFN y JSON-LD.

Generalmente Web Data Commons genera un nuevo dataset a finales de cada año por lo que a veces puede resultar obsoleto.

La información extraída se encuentra en un formato que no es conforme por completo con la especificación N-Quads, que se puede encontrar en https://www.w3.org/TR/n-quads/.

Esta especificación define líneas que contienen 4 valores que representan un sujeto, predicado, objeto y un grafo en ese orden.

En el caso de este dataset estos valores representan:

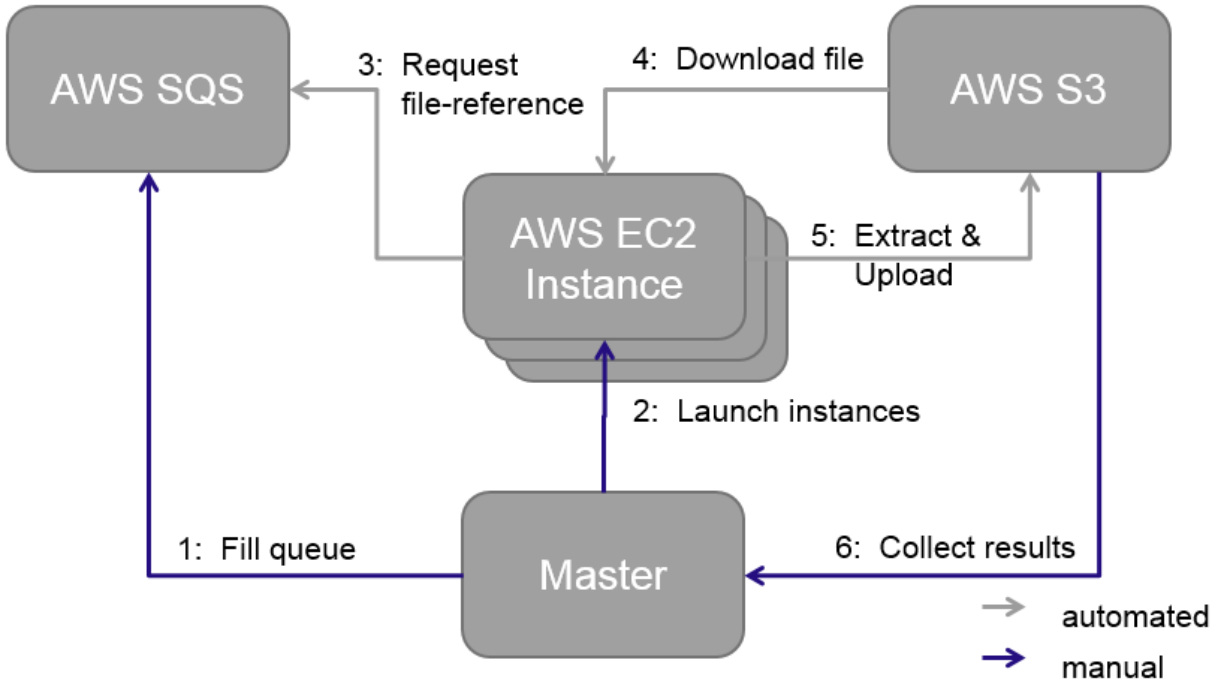
- El sujeto puede representar tanto la página web como un objeto integrado mediante una de las especificaciones disponibles.
- El predicado define el tipo de datos que se encuentra en el objeto. En este caso puede definir un objeto nuevo o una propiedad del actual.
- El objeto puede definir el tipo del nuevo objeto, el valor de una propiedad o una referencia a otro sujeto.
- El grafo representa la url de la página a la que pertenece la tupla.

Framework de extracción

El framework que proporciona WDC se trata de un proyecto Java Maven que se apoya en servicios de Amazon Web Services (AWS) para realizar una extracción de datos distribuida y paralela mediante instancias spot de EC2, almacenamiento en S3 y el uso de Simple Queue Service (SQS). Esto permite el rápido procesamiento de múltiples terabytes de datos de forma automática en cuestión de pocas horas.

El framework incluye por defecto cuatro procesadores distintos que serán explicados en detalle en un apartado posterior.

Funcionamiento general



La figura X.X es un esquema que representa las relaciones entre el nodo maestro (Master) y los distintos servicios de AWS utilizados por el framework.

A continuación, se desarrolla el funcionamiento de cada paso:

1. Desde el nodo maestro, se rellena una cola del "Servicio Simple de Colas" (Simple Queue Service | SQS) de AWS con todos los archivos que deben ser procesados. Estos archivos representan las tareas de los nodos de procesamiento y son referencias a archivos.
2. Desde el nodo maestro se lanza un número de instancias EC2 de AWS. Una vez lanzadas, cada instancia descargará automáticamente el framework y lo iniciará.
3. Cada instancia, al lanzar el framework, pedirá una tarea a la cola de SQS y empezará a procesar el archivo.

4. El archivo será descargado desde S3 y empezará a ser procesado.
5. Tras acabar cada archivo, el resultado será almacenado en un bucket de S3 definido por el usuario y se almacenarán estadísticas en AWS SimpleDB.
6. Cuando la cola esté vacía, el maestro puede empezar a recolectar la información extraída y las estadísticas.

Requisitos previos

- **Cuenta o Usuario de Amazon Web Services**: debido a que el framework está preparado actualmente para realizar el procesamiento utilizando servicios de Amazon, es necesario disponer de una cuenta de AWS con el número de créditos suficiente.
- **Token de acceso (access token) de AWS**: para acceder a los servicios de Amazon a través del framework es necesario contar con una clave de acceso (access key) de AWS así como de la clave secreta (secret asccess key). Ambas están disponibles a través de la web de AWS en la sección IAM (http://aws.amazon.com/iam/);
- **Java**: una máquina debe actuar como maestro. Esta máquina puede ser un ordenador local o una instancia de EC2 en AWS y debe tener instalada una versión superior a Java 6 (la versión actual corre sobre Java 8) para ejecutar los comandos y controlar la extracción.

  Debido a que por el momento la mayoría de versiones de Java han sido retiradas de los principales repositorios, la forma de adquirir Java es descargar una version desde la página oficial de Oracle.
- **Maven**: para poder compilar este framework se necesita una instalación de Maven 3. El uso de Maven 2 no es posible ya que ciertos añadidos está disponible únicamente a partir de la versión 3. Maven se puede instalar mediante el comando:

  ```
  apt-get install maven
  ```
- **Subversion**: para poder descargar el código del framework es necesario tener instalado subversion, una herramienta de revisión y versionado de software.

  ```
  apt-get install subversion
  ```

Archivo de configuración

El framework cuenta con un archivo con formato .properties que contiene atributos con los que se puede modificar el comportamiento del framework.

En la carpeta src/main/resource/ se encuentra una versión preliminar del archivo necesario llamado dpef.properties.dist. Es necesario crear una copia de este archivo en la misma carpeta con nombre: dpef.properties. Esto se puede llevar a cabo mediante la siguiente orden teniendo en cuenta que el directorio actual es la raíz del framework:

```
cp src/main/resource/dpef.properties.dist src/main/resource/dpef.properties
```

Los atributos que se pueden configurar en este archivo son:

- **awsAccessKey**: clave de acceso de AWS. Se puede encontrar en la sección IAM de la web de AWS. Este atributo no tiene valor por defecto.
- **awsAccessKey**: clave secreta de AWS. Se puede encontrar en la sección IAM de la web de AWS. Este atributo no tiene valor por defecto.
- **ec2keypair**: nombre del par de claves con el que se podrá acceder a la instancia en caso de ser necesario. La configuración de estas claves se puede realizar en la sección Key Pairs de AWS y estas son independientes de cada región por lo que se debe asegurar que existe un par de claves con el nombre especificado en la región indicada en el atributo ec2endpoint. Este atributo no tiene valor por defecto.
- **resultBucket**: nombre del bucket donde se van a almacenar los resultados de los archivos procesados. Este atributo no tiene valor por defecto. Este bucket estará ubicado en la región us-east-1 (N. Virginia) por defecto; si se quiere modificar este aspecto se debe añadir el atributo s3service.s3-endpoint al archivo jets3.properties ubicado en la misma carpeta.
- **deployBucket**: nombre del bucket donde se encontrará la versión del framework que deben ejecutar las instancias de EC2. Este atributo no tiene valor por defecto. Este bucket se encontrará en la misma región que el anterior.
- **dataBucket**: nombre del bucket de S3 en el que se encuentra el dataset con el que se desea trabajar. El valor por defecto de este atributo es "aws-publicdatasets". En caso de que el dataset que se va a utilizar pertenezca a Common Crawl, el valor de este atributo deberá ser "commoncrawl".
- **dataPrefix**: prefijo de los datos que se quieren extraer. Deberá ser el nombre de la carpeta que contenga los archivos a procesar dentro del bucket indicado. El valor por defecto de este atributo es" common-crawl/crawl-data" pero debe ser únicamente "crawl-data" en caso de que se quiera trabajar con el dataset de Common Crawl.
- **deployFilename**: nombre del archivo que será cargado al bucket señalado en *deployBucket* y que será descargado por las instancias de EC2. El valor por defecto de este atributo es "pdef.jar" y no hay ninguna necesidad de modificarlo.

- **ec2endpoint**: endpoint de la API de EC2 que se desea usar. Este atributo determina la región de AWS donde se van a lanzar las instancias de EC2. El valor por defecto de este atributo es "ec2.us-east-1.amazonaws.com"; esto indica que la región por defecto es us-east-1 (N. Virginia). Otros posibles valores para este campo se pueden encontrar en la documentación de AWS referente a Service Endpoints.

- **ec2ami**: identificador de la Amazon Machine Image (AMI) que se desea utilizar en las instancias de EC2. Esto indica el sistema operativo y los programas que van a tener las máquinas tras su creación.El valor por defecto de este atributo es ami-018c9568, que es una imagen de Ubuntu 14.04. Para el caso de este framework lo más indicado es seleccionar una AMI de Ubuntu u otro derivado de Debian. Se puede obtener más información sobre este sujeto en la documentación de AWS referente a Amazon Machine Images.

- **ec2instancetype**: nombre del tipo de instancia de EC2 que se desea lanzar. Dicho tipo definirá el número de procesadores, memoria RAM, ancho de banda, etc del que dispondrá la instancia. El valor por defecto de este atributo es c3.2xlarge. Este valor ya no es válido ya que el tipo de instancia indicado ya no está disponible. Los mejores tipos de instancia para realizar estas tareas serán aquellos que empiezan por C, lo que indica que están optimizados para procesos de computación. El usuario debe comprobar que los recursos asignados a la instancia (CPU, RAM) son suficientes para cumplir su función. Se puede obtener más información sobre los distintos tipos de instancias en la documentación de AWS relacionada con EC2 Instance Types y sobre sus precios en la sección Spot Instance Prices.

- **jobQueueName**: nombre de la cola de SQS en la que se van a guardar y en la que se van a leer las tareas a realizar por las instancias de EC2. El valor por defecto de este atributo es "jobs" y no ha necesidad de cambiarlo.

- **queueEndpoint**: endpoint de SQS que se desea utilizar. El valor por defecto de este atributo es "https://queue.amazonaws.com/" lo que indica que la región por defecto es us-east-1 (N. Virginia). Otros posibles valores para este campo se pueden encontrar en la documentación de AWS referente a Service Endpoints.

- **dataSuffix**: sufijo del tipo de archivos que se quieren procesar. El valor por defecto de este atributo es ".wat.gz". Otros posibles valores para este atributo son ".arc.gz", ".warc.gz"…

- **batchSize**: tamaño de los lotes de mensajes que rellenan la cola. El valor por defecto de este atributo es "10" y no hay necesidad de cambiarlo.

- **jobTimeLimit**: tiempo en segundos que la cola de SQS espera a que un mensaje que ha sido leído sea devuelto. Esto implica que si el mensaje no se elimina porque no se ha terminado de procesar adecuadamente se volverá a introducir en la cola para ser procesado de nuevo. Un buen valor para este atributo es tres veces el tiempo medio en que se tarda en procesar un archivo. El valor por defecto de este atributo es "900".

- **jobRetryLimit**: numero de veces que se reintenta procesar un mensaje determinado antes de dejarlode lado y escribir un error en las estadísticas. El valor por defecto de este atributo es "3" y no hay necesidad de cambiarlo.

- **sdbdatadomain**: nombre del dominio de AWS SimpleDB en el que se escribirán las estadísticas extraidas de cada archivo. El valor por defecto de este atributo es "data".
- **sdberrordomain**: nombre del dominio de AWS SimpleDB en el que se escribirán los errores ocurridos mientras se procesan los archivos. El valor por defecto de este atributo es "failures".
- **minResults**: numero mínimo de registros que debe haber en los dominios anteriores para que el framework descargue los datos. El valor por defecto de este atributo es "5".
- **processorClass**: clase que será utilizada para procesar los archivos. Es necesario que esta clase importe "org.webdatacommons.framework.processor.FileProcessor". El valor por defecto de este atributo es "org.webdatacommons.hyperlinkgraph.processor.WatProcessor".
- **javamemory**: memoria que se asignará mediante la opción -Xmx al proceso Java que se lance en cada instancia. El valor por defecto de este atributo es "5G".

Los siguientes atributos son utilizados por los extractores de ejemplo y no tienen relevancia en caso de que se vaya a crear uno nuevo.

logRegexFailures: booleano que determina si se registra un error en caso de que se detecte un falso positivo en el extractor de datos estructurados. El valor por defecto de este atributo es "false";

extracTtopNTerms: en la versión actual no tiene ningún uso.

extractionAlgorithm: clase que el procesador utilizará para extraer los datos que se desean. El valor por defecto de este atributo es "org.webdatacommons.webtables.extraction.BasicExtraction";

phase1ModelPath y phase2ModelPath: indican modelos que determinan el tipo de tabla que se está extrayendo. Los valores por defecto de estos atributos son "/RandomForest_P1.mdl" y "/RandomForest_P2.md" (en la versión actual el segundo atributo acaba en .md cuando debería acabar en .mdl).

Comandos

Queue

Añade mensajes a la cola. Estos mensajes son referencias a archivos que van a ser procesados.

Parámetros

- **bucket-prefix o p**: indica el comienzo de la ruta de los archivos que se quieren añadir a la cola. Esta ruta debe excluir el prefijo indicado en la configuración dataPrefix. Un ejemplo de este parámetro es: CC-MAIN-2016-44/segments/1476988717783.68/warc/. La lista de posibles valores se encuentra en los archivos .paths explicados en el apartado Common Crawl.
- **file-number-limit o l**: indica el número máximo de archivos que se deben añadir a la cola.
- **bucket-prefix-file o f**: indica la ruta a un archivo con una lista de prefijos que se desean añadir a la cola.

Funcionamiento

El comando fallará en caso de que no se proporcione ni un prefijo ni un archivo con una lista de prefijos.

Para cada uno de los prefijos seleccionados, se listarán todos los elementos del bucket dentro de la carpeta dataPrefix cuya ruta empiece por el prefijo indicado y se extraerá su ruta completa, llamada key. Estos mensajes se enviarán por lotes del tamaño indicado en batchSize. Se dejarán de enviar mensajes cuando alcance el límite de archivos indicado o cuando se hayan enviado todos los archivos de todos los prefijos indicados.

Clearqueue

Elimina la cola de SQS y todos los mensajes que contiene.

Deploy

Parámetros

- **jarfile o j**: indica la ruta del archivo .jar a subir.

Funcionamiento

Sube el jarfile indicado al deployBucket con el nombre deployFilename indicado y permite acceso de lectura en el bucket a todos los usuarios.

Start

Parámetros

- **worker-amount o a**: cantidad de instancias de EC2 a lanzar.
- **pricelimit o p**: límite de precio en dólares de las instancias de EC2 que se van a lanzar.

Funcionamiento

Manda una petición para lanzar el número de **instancias spot** de EC2 indicado con el precio máximo señalado. Esto significa que se van a utilizar máquinas que pueden ser interrumpidas en cualquier momento bien porque el precio de esta supera el límite establecido o porque disminuye la oferta de instancias spot a cambio de una gran reducción en el precio por hora en comparación con las estancias EC2 estándar.

Las instancias creadas serán del tipo ec2instancetype, con una ami ec2ami y con el par de claves ec2keypair.

Finalmente se añadirá un script de shell que será ejecutado por el usuario root en cuanto se lance la instancia. Este script será distinto en función del gestor de paquetes que utilice el sistema operativo y de las herramientas que la ami seleccionada tenga instaladas por defecto.

Los scripts incluidos en la versión actual del framework no son válidos ya que intentan descargar una versión de Java que ya no está disponible en los repositorios principales.

El script más completo para una ami de Ubuntu que no incluye Java constaría de las siguientes órdenes:

```
#!/bin/bash
echo 1 > /proc/sys/vm/overcommit_memory
apt update
apt install openjdk-8-jdk
wget -O /tmp/start.jar http://s3.amazonaws.com/{deployBucket}/{deployFilename}
java -Xmx {javamemory} -jar /tmp/start.jar > /tmp/start.log &
```

A continuación, se explica el funcionamiento de cada línea de este script:

1. Indica que intérprete debe procesar el documento, en este caso bash.
2. Permite al kernel de Linux conceder más memoria RAM de la que está disponible a las aplicaciones que la pidan.
3. Actualiza la lista de directorios del gestor de paquetes.
4. Descarga e instala el openjdk de Java 8.
5. Descarga el framework cargado en el deployBucket.
6. Ejecuta el framework con la opción -Xmx indicada y escribe su salida en el archivo log indicado.

Monitor

Muestra por consola el estado de la extracción.

Parámetros

- **autoShutdown o a**: si está presente, una vez que se acaben los mensajes de la cola, se detendrán las instancias de EC2 lanzadas.

Funcionamiento

En un bucle, primero se calculan el número de instancias que se han solicitado y cuantas de esas están disponibles observando si el estado de las peticiones es activa o abierta.

Después se calcula el número de mensajes que hay en la cola y cuántos de ellos se encuentran "en vuelo", lo que significa que han sido leídos y deberían estar procesándose.

En caso de que esté presente la opción autoShutdown, si hay instancias activas y la cola está vacía se incrementa un contador con el tiempo que haya transcurrido. Si la cola no está vacía ese contador se reinicia. Si el contador supera un cierto umbral, se detienen todas las instancias.

Cada segundo se crea un registro con el tiempo y la longitud de la cola y se añade a una lista. Si un elemento de esa lista tiene una marca de tiempo anterior a un cierto valor se elimina. Finalmente se comparan el primer y último elemento de esa lista y se calcula el tiempo estimado que falta para el final de la extracción.

Por último, se escribe todo en la consola y se vuelve al primer paso.

Shutdown

Lista las instancias spot con estado "activo" y las detiene.

Cleardata

Parámetros

- **includeS3Storage o r**: si está presente también se eliminará la informcion almacenada en el reultBucket.

Funcionamiento

Se listan todos los dominios existentes en SimpleDB y para cada uno se lanza un hilo que lo elimina. Una vez terminada esta tarea, si la opción includeS3Storage está presente, se eliminarán los archivos resultantes de la extracción del bucket.

Crawlstats

Parámetros

- **bucket-prefix o p**: inicio de la ruta de los archivos cuyas estadísticas se quieren calcular.
- **output-file o o**: ruta del archivo donde se va a escribir la salida de este comando.

Funcionamiento

Actualmente este comando no parece funcionar adecuadamente.

Retrievedata

Parámetros

- **destination o d**: directorio en el que se van a escribir los archivos descargados.
- **localsource o l**: ruta del directorio local desde donde se quieren extraer la información.
- **multiThreadMode o m**: número de hilos utilizados para esta tarea.

Funcionamiento

Si no se ha proporcionado un valor para localsource, se descargarán los archivos extraidos en la carpeta data del resultBucket.

En caso de que si se haya indicado este valor, se recorrerán todos los archivos que contenga ese directorio y se sumarán.

DESAROLLO

1 AMEF

El resultado final de este proyecto ha recibido el nombre de AMEF que proviene de las siglas de framework de extracción de metadatos de accesibilidad (Accessibility Metadata Extraction Framework). El objetivo de esta herramienta es extraer la mayor cantidad de información posible sobre los metadatos relacionados con accesibilidad de la forma más óptima y precisa posible a partir de los esquemas proporcionados por schema.org.

La herramienta desarrollada se basa en el framework proporcionado por Web Data Commons, que ha sido modificado para permitir el procesamiento tanto de forma distribuida en la nube como en una máquina local.

El formato de datos extraído está orientado a su almacenamiento en MongoDB con el fin de poder realizar consultas complejas y extraer la información que el usuario desee.

Se ha desarrollado una interfaz web para poder visualizar una serie de consultas básicas sobre los datos extraídos.

2 Tecnologías utilizadas

Procesamiento

Software

Para la herramienta que se encarga del procesamiento, manejo y análisis de los datos se ha utilizado **Java**, en concreto la versión **1.8** ya que es el lenguaje en el que está desarrollado el framework original en el que se ha basado esta herramienta.

Se ha utilizado Maven como una herramienta de gestión de dependencias, las cuales son las siguientes librerías:

- **Mongo-java-driver**: gestiona la comunicación entre el programa y una base de datos MongoDB.
- **Heritrix-commons**: contiene varias librerías de servicios generales de Java.
- **Gson**: análisis, lectura y escritura de formato JSON.
- **Jsoup**: librería que se encarga del análisis y lectura de HTML.
- **Aws-java-sdk y jets3t**: proporcionan métodos para interactuar con distintos servicios de AWS.

- **Slf4j-log4j12 y log4j**: proporcionan formas de logging y registros de errores.
- **Jsap**: librería que facilita la creación de interfaces de línea de comandos.
- **Jwat** (common, gzip y warc): proporciona herramientas para la lectura, escritura y validación de archivos WARC, ARC y GZIP.
- **Commons-configuration**: lectura y escritura de archivos de configuración.

Hardware

El procesamiento de los datos se puede realizar de forma local pero también se puede realizar mediante el uso del servicio **EC2 (Elastic Compute Cloud)** de Amazon Web Services, que proporciona capacidad informática en la nube segura y de tamaño modificable.

Esta herramienta hace uso de instancias spot, lo que significa que se conseguirá una gran reducción del precio respecto a las instancias bajo demanda a cambio de la posibilidad de que la máquina sea detenida si así lo requiere la disponibilidad.

Almacenamiento

Para el almacenamiento de los datos se utilizan dos herramientas:

- **MongoDB**: es una base de datos NoSQL de código abierto. Esto permite almacenar datos semiestructurados de forma flexible y sin fuertes relaciones. Además, está diseñado para ofrecer alta disponibilidad y escalabilidad lo cual puede ser útil si se están procesando muchos archivos al mismo tiempo.
- **S3 (Simple Storage Service)**: es un servicio de Amazon Web Services que ofrece escalabilidad, disponibilidad de datos, seguridad y rendimiento. Es útil tanto para almacenar el software necesario para ser descargado por las instancias de EC2 como para almacenar los resultados del análisis.

Web

Para el desarrollo de la interfaz web se ha utilizado Angular, un framework de código abierto desarrollado por Google basado en typescript (una ampliación de JavaScript) que permite crear aplicaciones dinámicas basadas en componentes.

Al ser una herramienta muy extendida, cuenta con muchas librerías desarrolladas por terceros como nvd3 y d3 que en este caso usaremos para facilitar la visualización de los datos extraídos.

3 Procesamiento local y en la nube

Con el objetivo de que esta herramienta permita tanto el procesamiento mediante el uso de Amazon Web Services como independiente a este servicio, se han generado un sistema de colas y de almacenamiento que se adaptan al entorno en el que se está llevando a cabo la tarea.

Sistema de colas

En caso de que se esté llevando a cabo el procesamiento en instancias EC2 de AWS, se utilizará el servicio de SQS para determinar los archivos que se desean extraer mientras que si se está realizando de forma local se utilizará un nuevo sistema explicado a continuación.

El sistema de colas diseñado para el procesamiento local utiliza MongoDB de forma que proporciona un funcionamiento prácticamente idéntico al que utiliza el framework original en un aspecto: SQS permite definir un límite de tiempo en el que los mensajes volverán a estar disponibles tras haberse leído si no son borrados. En la herramienta desarrollada no se hace uso de esta utilidad.

Este sistema se ha implementado utilizando un patrón adaptador por lo que el código original no cambia de forma drástica.

Las colas de mensajes son colecciones de MongoDB generadas dentro de una base de datos llamada **queues.** Los mensajes serán documentos de esta colección y están compuestos de dos campos:

- **Body**: será una cadena que determina el contenido del mensaje.
- **State**: será un valor 0 o 1 que representará si el mensaje está disponible o ya ha sido leído respectivamente.

Para añadir mensajes a la cola, se hará de la misma forma que en SQS; los mensajes a enviar se introducirán en un lote y cuando este alcance el tamaño determinado por el usuario, se introducirán los mensajes que contenga y se vaciará y se continuarán añadiendo mensajes.

También permite la limpieza de la cola, lo que eliminará todos los documentos que contenga la colección.

Con respecto a la lectura de mensajes, para evitar que más de un hilo reciba el mismo mensaje se utiliza el método de MongoDB que actualiza y devuelve un elemento simultáneamente, estableciendo su valor en no disponible para otros hilos antes de leerlo.

Almacenamiento

En caso de que se lleve a cabo el procesamiento de forma local, los datos extraídos serán introducidos directamente en la base de datos ubicada en la dirección indicada en el archivo de configuración.

En caso de que se realice en AWS, los datos serán escritos en buckets de S3. Se escribirán dos archivos por cada uno procesado: uno en una ruta stats, que contendrá las estadísticas de la extracción y otro en la

ruta pages, que contendrá una lista de las páginas en las que se han encontrado metadatos con toda la información recogida sobre estas.

Esta información se encontrará en formato JSON para poder ser introducida más tarde en MongoDB.

4 Formato de datos

La salida de datos de esta herramienta se encuentra en formato bson ya que es el aceptado por MongoDB.

Cada archivo procesado generará una lista de documentos que se considerará la información sobre metadatos extraída de este y otro documento que almacenará estadísticas generadas durante el procesamiento de dicho archivo.

Estadísticas

Los documentos de estadísticas se encontrarán en una colección llamada **stats** y contendrán los siguientes campos:

- **File**: indica el archivo al que hace referencia las estadísticas que siguen.
- **Date**: fecha en la que se creó el dataset sobre el cual se ha llevado a cabo el análisis con esta herramienta.
- **Total**: número total de páginas web distintas que se han encontrado en el archivo procesado.
- **Found**: número de páginas web del archivo en las que se ha encontrado al menos un elemento de metadatos de accesibilidad.
- **Duration**: tiempo que ha tardado en procesarse el archivo.
- **Rate**: número de páginas por segundo procesadas.

Páginas

La información extraída se almacenará en una colección llamada **pages** en la que cada documento representará una página web en la que se han encontrado metadatos de accesibilidad. El formato de cada uno de estos documentos será el siguiente:

- **File**: archivo en el que se ha encontrado la página web a la que hace referencia este documento.
- **Date**: fecha en la que se creó el dataset sobre el cual se ha llevado a cabo el análisis con esta herramienta.
- **Url**: enlace de la página web a la que hace referencia este documento.

- **Domain**: dominio de la página web.
- **Items**: lista formada por objetos que representan elementos definidos por schema.org. Cada uno de estos objetos está formado por los siguientes valores:
    - **Format**: especificación de integración de metadatos con la que está representado este elemento. Puede ser uno de los siguientes valores: Microdata, RDFa y JSON-LD.
    - **Name**: nombre del objeto (CreativeWork, Book…). Si el valor de este campo es *none*, significa que se han encontrado metadatos en un contexto en el que no hay un objeto declarado.
    - **Tag**?: Etiqueta de HTML en la que está definido este objeto. Dependiendo del extractor utilizado este campo puede no aparecer. Puede ser *none* en las mismas condiciones que el campo anterior.
    - **Properties**: lista de metadatos encontrados en el objeto definidos por los valores:
        - **Name**: tipo de propiedad (accessibilityFeature, accessibilityControl…)
        - **Value**: valor de la propiedad (highContrast, fullKeyboardControl…)
        - **Section**?: sección (head o body) en la que se ha encontrado este metadato. Solamente aparece si el campo Name del objeto (no el de esta propiedad) es *none* o *html*.

5 Extractores

WAT (Rechazado)

Tras estudiar los distintos datasets disponibles para el análisis ofrecido por Common Crawl se escogió el formato WAT para desarrollar un extractor ya que este parecía el más óptimo gracias a su formato JSON que contiene información sobre los metadatos de una página en un tamaño de archivos muy inferior a aquel del formato WARC.

Durante el desarrollo de este se llegó a la conclusión de que los resultados que podría proporcionar este método serían muy inexactos ya que, en este formato, la información de los metadatos almacenada no incluye ciertos atributos necesarios para la correcta identificación de las propiedades definidas por schema.org.

Debido a que, por ejemplo, el campo itemprop de los metadatos, pieza fundamental de la especificación de Microdata, no está almacenado en estos archivos es imposible conocer con seguridad a que propiedad hace referencia, confundiéndose los valores de accessModeSufficient con algunos de accessMode u otros valores como none, index, transcript… con otras propiedades sin ninguna relación.

Por este motivo se rechazó la idea de trabajar con este formato e intentar conseguir resultados con un formato más completo a pesar del mayor coste de procesamiento que supone.

WARC

Este extractor analiza **archivos de formato WARC** de la forma más completa y precisa que se puede conseguir con la información disponible. Esta precisión se consigue a cambio de una necesidad de capacidad de procesamiento considerablemente mayor a la de los demás métodos considerados.

Este extractor consta de dos clases llamadas amef.processor.SchemaWarcProcessor y amef.processor.SchemaHTMLExtractor.

La primera, SchemaWarcProcessor, se encarga de separar los contenidos del archivo en WarcRecord o registro WARC, en los que se dividen estos archivos y de admitir para ser procesados únicamente aquellos que son de tipo response.

De estos registros se extrae tanto su contenido en bytes como la url de la página web que contiene.

Este contenido en bytes se analiza para crear un objeto document de jsoup (una librería java dedicada al análisis de HTML) y este se procesa en la clase de extracción mencionada.

Para llevar a cabo la extracción de metadatos se hace un recorrido en profundidad del documento llamando a un método llamado head la primera vez que se encuentra un nodo y a otro método llamado tail cuando se abandona.

Para saber en qué contexto se encuentra el nodo actual se mantienen 3 pilas: una con objetos representados en microdata, otra con objetos representados en RDFa y finalmente una con vocabularios RDFa.

Cuando se llama al método head se comprueba si se declara un objeto de microdata o RDFa o un vocabulario y si coincide con las características deseadas (generalmente que pertenezca a schema.org), se añade a la pila correspondiente. Si ese nodo contiene una propiedad de metadato de las que se está buscando, se añadirá al último objeto que se haya introducido en la pila. Si el nodo es un script de tipo json+ld, se analizará en busca de metadatos. Finalmente, si el nodo tiene etiqueta HTML y no se ha introducido un objeto en alguna de las pilas, se introducirá uno con nombre *none* para que siempre haya al menos uno en la pila ya que puede ocurrir que se encuentren metadatos antes de la declaración de un objeto.

Cuando se llama al método tail se comprueba si en la llamada a head de ese nodo se ha introducido un objeto en alguna de las pilas, y en caso positivo, se extrae ese objeto y se guarda en una lista si se han encontrado metadatos. También se comprueba si se ha introducido un vocabulario en la pila y se elimina.

Cuando se acaba de recorrer el documento se devuelve una lista con los objetos encontrados con sus correspondientes metadatos.

Finalmente se genera el documento con las estadísticas de la extracción y se almacena según el método correspondiente.

WDC

Finalmente se ha desarrollado un extractor sobre el **dataset de metadatos que ofrece Web Data Commons**. El análisis de este dataset requiere de menos capacidad de procesamiento que el anterior, pero a cambio los resultados pueden ser más inexactos.

En este extractor, se lee cada línea, se extraen los 4 fragmentos de información que contiene y se procesa.

Si la url a la que hace referencia no es la misma que la considerada actual, se sustituirá por la nueva. Si se han encontrado metadatos en esa página anterior, se añadirá a una lista de documentos y se vaciará la lista de objetos encontrados.

Si el nodo declara un nuevo objeto de schema.org, se añade un nuevo objeto a esta lista. Si se encuentra una referencia a un objeto que no se encuentra en la lista, se introducirá un objeto de nombre *none*.

Finalmente, si el tipo de información es una propiedad de schema.org y el valor coincide con uno de los valores aceptados, se añadirá al objeto adecuado.

Cuando se han leído todas las líneas, se genera un documento con las estadísticas y se almacena según el método correspondiente.

Este método encuentra ciertos inconvenientes ya que se ha observado que los metadatos que se encuentran fuera de un objeto no son recogidos o que el origen de las propiedades puede no estar bien indicado.

Además, existe un fallo de la máquina virtual de Java que provoca que ciertos archivos de forma aleatoria produzcan un error de final de archivo inesperado al ser procesados mediante GZIPInputStream.

Debido a que para aquellos metadatos que no contengan el atributo content, se almacenará todo el texto que se encuentre en los elementos siguientes, para utilizar este extractor es recomendable definir los valores que se desean aceptar en el archivo search.txt.

6 Configuración

Archivo de definición de búsqueda

Este archivo, ubicado en src/main/resources/search.txt permite al usuario definir las propiedades y valores de estas que se desean extraer. El archivo tiene la siguiente estructura:

Cada propiedad debe encontrarse en una línea distinta. Al comienzo de la línea se debe encontrar el nombre de la propiedad que se desea encontrar. En caso de aceptar cualquier valor la línea debe contener únicamente este parámetro. Si se desean definir los valores aceptados para esta propiedad, se deben introducir justo después de esta dos puntos y a continuación una lista, separada por comas, de los valores aceptados, todo ello sin espacios.

Archivo de propiedades

Este archivo se encuentra en src/main/resources/amef.properties y es muy similar al proporcionado por el framework original excepto por algunas adiciones:

- QueueService: determina el servicio de colas que se desea utilizar. Sus valores pueden ser sqs o mongo.
- StorageService: determina donde se van a escribir los datos extraídos. Sus valores pueden ser s3 o mongo.
- FileOrigin: determina si los archivos que se desean procesar se encuentran en un bucket de S3 o se descargan desde una URL.
- MongoUrl: ip o dominio de la instancia de MongoDB que se desea utilizar.
- MongoPort: puerto en el que se puede acceder a la instancia de mongo en la url anterior.
- MongoDB: nombre de la base de datos en la que se quieren almacenar los resultados.
- ParallelFiles: número máximo de hilos paralelos que se desean lanzar. Si supera al número de procesadores de que dispone la máquina, el límite será este último.

MANUAL DE USUARIO

Instalación inicial

En este apartado se describen los pasos a seguir para poder hacer uso de la herramienta desarrollada:

- Instalación de Java versión 1.8.
- Instalación de Maven 3.
- En la máquina donde se desee almacenar los datos resultantes y en aquella en la que se quiera establecer el servicio de colas es necesaria una instalación de MongoDB.
- Acceso a una cuenta de AWS con créditos suficientes. Acceso a claves de acceso y secreta.
- Si el procesamiento se va a ejecutar en instancias de EC2 es necesario crear un bucket en el servicio S3 para almacenar el software que debe ser descargado por las instancias de EC2.
- Si se van a escribir los resultados en el servicio S3 es necesario crear otro bucket.
- Si se va a usar la interfaz web de visualización de datos se deben instalar tanto Node.js como Angular Cli.
- Si la instancia de MongoDB que se desea utilizar no se encuentra en la misma máquina que se está ejecutando la herramienta, es necesario modificar el archivo de configuración mongod.cfg y eliminar la configuración bindIp o añadir aquellas IPs desde las que se desea permitir el acceso y abrir el puerto al que se desea acceder. Si se ha optado por eliminar el campo es necesario tener en cuenta que la base de datos está expuesta a cualquier acceso y actualmente no está soportada la autentificación de mongo en la herramienta.

Ejecución del procesador

Configuración

Antes de empezar a ejecutar la herramienta se deben establecer ciertos valores en los archivos de configuración.

En el archivo amef.properties que se encuentra en src/main/resources/ se deben establecer los siguientes parámetros:

- Necesarios siempre: queueService, storageService, fileOrigin, processorClass, parallelFiles, jobQueueName, batchSize, jobTimeLimit, dataSuffix.
- Necesarios si se va a utilizar algún servicio de AWS (SQS, EC2 o S3): awsAccessKey, awsSecretKey.

- Si se van a utilizar instancias de EC2: deployFilename, deployBucket, ec2keypair, ec2endpoint, ec2ami, ec2instancetype, javamemory.
- Si se va a utilizar MongoDB (como cola o como almacenamiento): mongoUrl, mongoPort, mongoDB.
- Si se va a utilizar S3 como almacenamiento de resultados: resultBucket.
- Si se va a utilizar un dataset contenido en S3: dataBucket, dataPrefix.
- Si se va a utilizar SQS como servicio de colas: queueEndpoint.

Una vez se haya acabado de establecer la configuración deseada se debe compilar el proyecto ejecutando el siguiente comando desde el directorio raíz:

```
mvn package
```

Ejecución de commandos

En la carpeta bin existen 2 archivos master y master.bat que permiten facilitar la ejecución de los comandos que se deseen. Por lo tanto, para ejecutar cualquiera de los comandos que se van a explicar más adelante se puede hacer ejecutando estos archivos de la siguiente forma si nos encontramos en la carpeta raíz:

```
./bin/master (en caso de Linux)
.\bin\master.bat (en Windows)
```

Encolado de archivos

Existen dos formas de añadir archivos a la cola.

El primer método solo es viable si los archivos que se desean añadir se encuentran en un bucket de AWS. En este método se debe llamar al comando queue con un parámetro -p en el que se indique el inicio de la ruta de los archivos que se quieren añadir, excluyendo el *dataPrefix* indicado en la configuración. Por ejemplo:

```
./bin/master queue -p CC-MAIN-2019-30/segments/1563195523840.34/warc/
```

El segundo método consiste en proporcionar un archivo, con el parámetro -f, en el que se encuentren todas las rutas enteras de los archivos que se deseen añadir. Este método ha sido modificado respecto al framework original para permitir una mayor eficiencia a la hora de introducir mensajes a la cola por lo que estas rutas se añadirán, aunque sean incorrectas. Este método también posibilita el hecho de que los archivos de rutas (.paths y .list) proporcionados por Common Crawl y Web Data Commons cumplen exactamente los requisitos de este comando.

Estos archivos se pueden obtener en https://commoncrawl.org/the-data/get-started/ y http://webdatacommons.org/structureddata/index.html respectivamente.

Un ejemplo sería:

```
./bin/master queue -f html-microdata.list -l 100
```

Este commando, con el parámetro -l añadirá las 100 primeras líneas del archivo indicado a la cola.

También existe la posibilidad de eliminar todos los mensajes de la cola mediante el comando:

```
./bin/master clearqueue
```

Lanzamiento del procesador

Existen dos comandos para empezar a ejecutar el procesador:

El primero es **run**. Este comando no requiere ningún parámetro adicional y da comienzo a la ejecución del extractor en la misma máquina. Debido a la cantidad de mensajes que genera este proceso es recomendable redirigir su salida a un archivo de log. Por ejemplo:

```
./bin/master run > amef.log 2> amef.error
```

El segundo es **launch**, lo que provoca que se lancen tantas instancias de EC2 como se indique con el parámetro -a y con un precio máximo que se indique con el parámetro -p.

Además, si se incluye el parámetro -j, antes de lanzarse estas instancias, se subirá el archivo jar que se indique al bucket de S3 desde el que los descargarán estas instancias por lo que es necesario incluirlo al menos la primera vez siempre que se modifique la configuración y se compile el proyecto.

Un ejemplo de este comando es:

```
./bin/master launch -p 1 -a 6 -j target/amef-1.0.0.jar
```

En caso de haber utilizado este último comando y se desea cancelar el procesamiento, se debe utilizar el comando **shutdown**, que cancelará todas las instancias que se hayan requerido.

Monitorización del proceso

Mediante el comando monitor se puede obtener información sobre el estado del proceso.

Esto mostrará por pantalla el número de mensajes que quedan en la cola, el número de mensajes que están siendo procesados, el ritmo al que están siendo procesados los archivos y el tiempo estimado que falta para que finalice.

En caso de que se esté ejecutando en instancias de EC2, también se indicarán el número de instancias que se han pedido y el número de ellas que se encuentran concedidas en el momento.

En caso de que se indique el parámetro -a, en el momento en el que se acaben los mensajes de la cola, se cancelarán las instancias de EC2 requeridas por la cuenta indicada por lo que se debe tener **cuidado** en caso de que haya instancias ajenas a este proceso corriendo al mismo tiempo.

Un ejemplo de este comando es:

```
./bin/master monitor (-a)
```

Recolección de datos

En caso de que se haya escogido mongo como servicio de almacenamiento de los resultados, la información ya se encontrará en la instancia de MongoDB deseada.

En caso de que se haya seleccionado S3, los datos se encontrarán en archivos de este servicio por lo que podremos utilizar el comando retrievedata.

Si se llama a este comando con el parámetro -l, se imprimirá una lista con todas las fechas disponibles para descargar.

Si se llama a este comando con el parámetro -d, se descargará la información contenida en los archivos correspondientes a esa fecha y se introducirá en la base de datos.

```
./bin/master retrievedata -l

./bin/master retrievedata -d 2019-07
```

Si se desean borrar datos almacenados tanto en S3 como en MongoDB se puede utilizar el comando deletedata.

Los parámetros que acepta este comando son -s y -m indicando de qué lugar se quieren eliminar los datos representando -s a S3 y -m a MongoDB. Ambos pueden incluirse juntos, pero siempre debe aparecer al menos uno. El otro parámetro que acepta es -d donde se debe indicar la fecha de los datos que se quieren eliminar. Si este parámetro no aparece, se eliminarán todos los datos

```
./bin/master deletedata -a -m

./bin/master deletedata -a -d 2019-06
```

Visualización de estadísticas

Se ha desarrollado una interfaz web simple en la que se pueden visualizar algunos de los datos más básicos y a la vez más útiles. Para realizar consultas más complejas se deben realizar directamente sobre MongoDB ya sea a través de la línea de comandos o de una interfaz gráfica.

Interfaz Web

Para poder ejecutar este servidor es necesario, como ya se ha indicado antes, que estén instalados Node.js y Angular Cli.

Primero se deben modificar el archivo src/enviroments/environment.ts y establecer el valor adecuado para la propiedad apiBaseUrl, que representa la dirección en la que se ubica el servidor de backend y el archivo backend/environment.ts y cambiar el valor de la propiedad database al nombre de la base de datos sobre la que se desea consultar la información.

Desde el directorio raíz se deben instalar las dependencias de angular necesarias mediante el comando:

```
npm install
```

Y ejecutar el servidor mediante el comando:

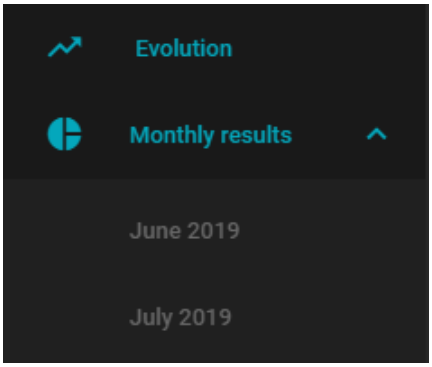
```
ng serve
```

Después se debe entrar en la carpeta backend e iniciar este servicio con los comandos:

```
npm install
npm start
```

Una vez que se han iniciado los dos servicios, se puede acceder a ellos en la dirección localhost:4200.

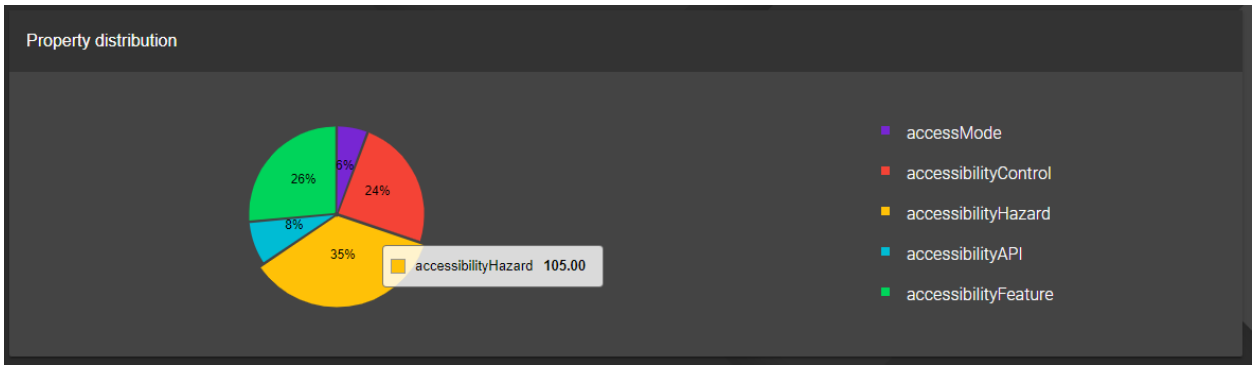
Esta interfaz consta de dos secciones a las que se puede acceder desde el menú situado en la parte izquierda de la pantalla.



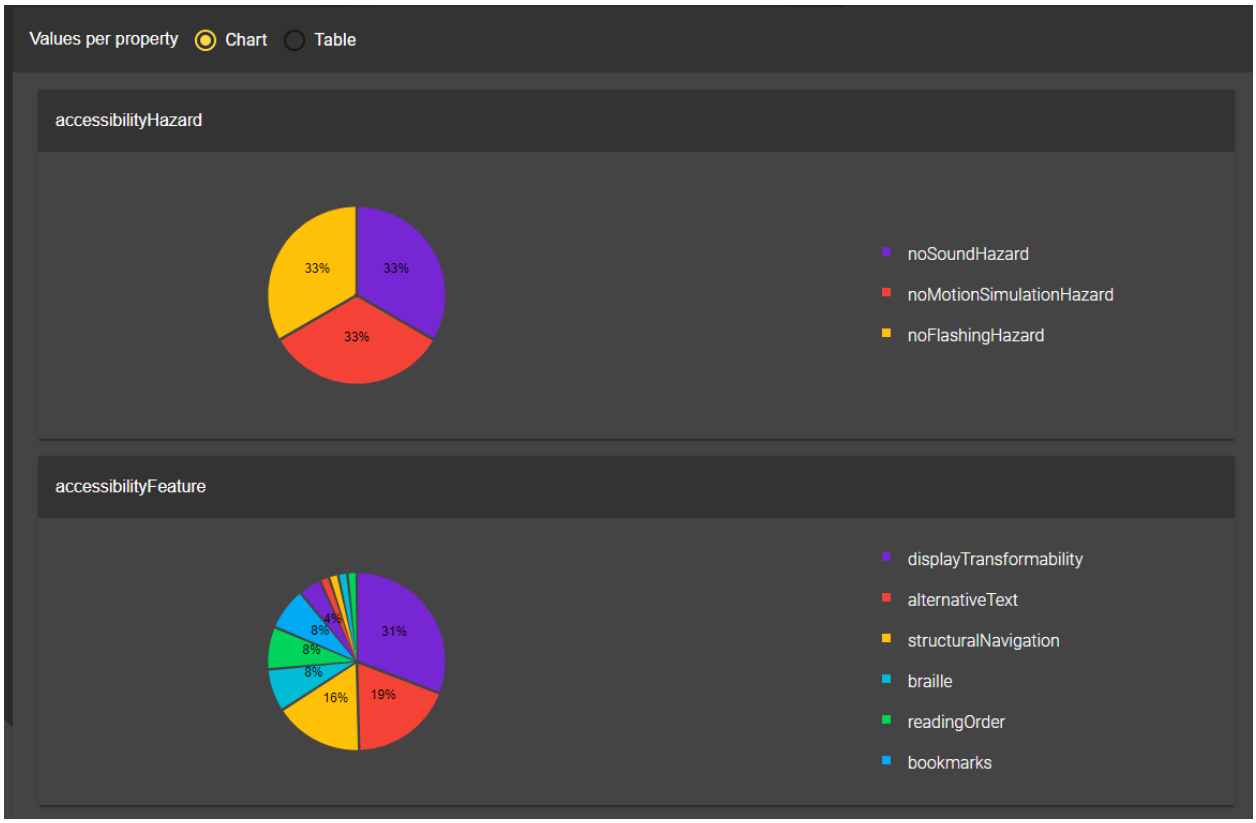
En la primera se puede observar la evolución de la cantidad de metadatos encontrados a lo largo de los distintos datasets a través del tiempo tanto en cantidades totales como en relación al total de páginas procesadas.

En la segunda se pueden observar datos sobre la extracción de cada fecha. Los datos que se pueden visualizar son:

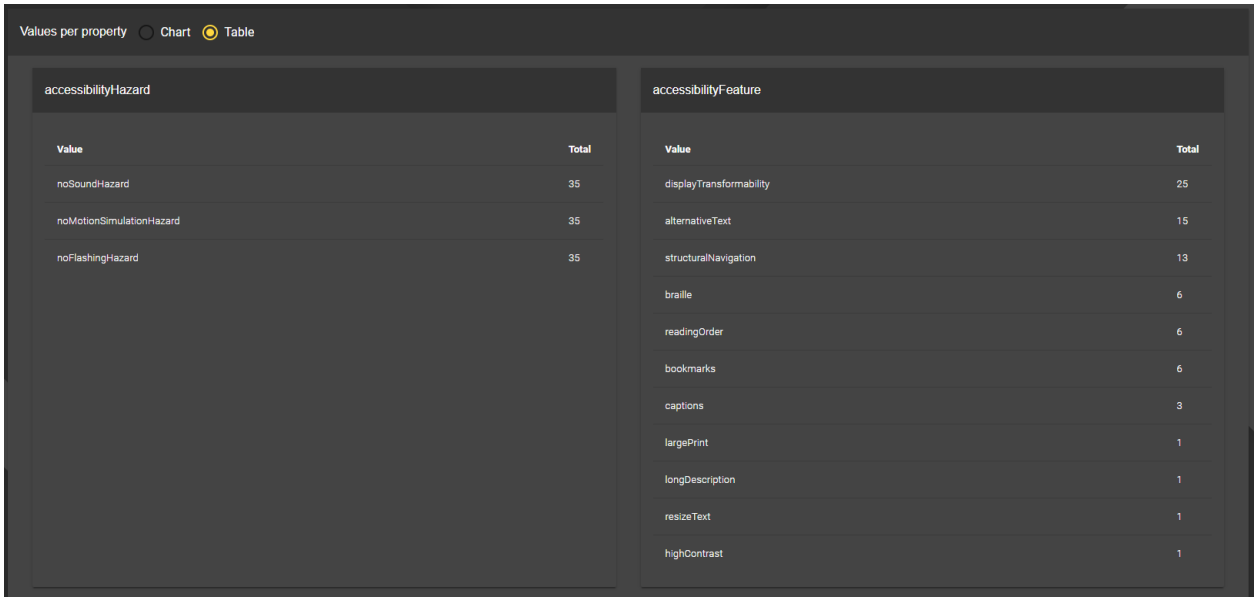
- Porcentaje de aparición de cada propiedad frente al total:



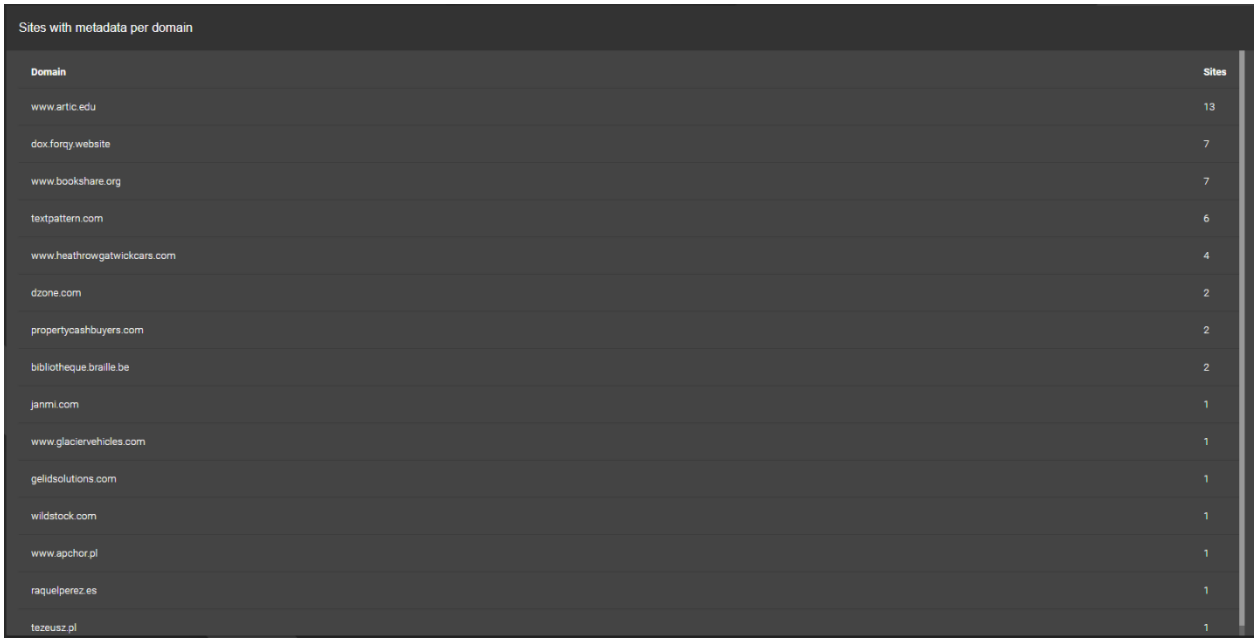
- Distribución de apariciones de cada valor para cada propiedad:
    - Tanto en gráfico:



    - Como en tabla:



- El número de páginas por dominio que contienen metadatos:



Consultas sobre MongoDB

La forma de conseguir extraer información compleja sobre los datos extraídos es realizar consultas sobre la base de datos directamente.

A continuación, se van a mostrar algunas consultas de ejemplo, incluyendo las que han sido utilizadas en la interfaz web:

- Total de páginas web procesadas y cantidad de estas en las que se han encontrado metadatos por cada fecha:

```
db.getCollection("stats").aggregate( [
        {$group: { _id: "$date", searched : {$sum: '$total'}, found : {$sum: '$found'}}},
        {$project: { date : '$_id', _id : 0, searched :1, found:1}},
        {$sort : {date : 1}}
    ])
```

- Cantidad de metadatos de cada propiedad encontrados en una fecha (en este caso 2019-07):

```
db.pages.aggregate( [
        {$match: {date : "2019-07"}},
        {$unwind: '$items'},
```

```
        {$unwind: '$items.properties'},
        {$group: {_id: '$items.properties.name', total: {$sum: 1}}},
        {$project: {key : '$_id' , total : 1, _id : 0}},
        {$sort: {property: -1}}])
```

- Lista de las propiedades encontradas con una lista con la cantidad de veces que se ha encontrado cada uno de sus valores en una fecha (2019-07)

```
db.pages.aggregate( [
        {$match: {date: "2019-07"}},
        {$unwind: '$items'},
        {$unwind: '$items.properties'},
        {$group: {_id: '$items.properties.name', value: {$push:"$items.properties.value"}}},
        {$unwind: '$value'},
        {$group: { _id: {name:"$_id",value:{$substr: ['$value', 0, {$indexOfBytes: ["$value", "/"]}]}},
total: {$sum:1}}},
        {$sort: {total: -1}},
        {$group: { _id: '$_id.name', values : {$push: {key: '$_id.value', total: '$total'}}}},
        {$project: {property: '$_id', values: 1, _id:0}},
        {$sort: {property: -1}}
    ])
```

- Lista con la cantidad de páginas con metadatos por dominio en una fecha determinada (2019-07):

```
db.pages.aggregate( [
        {$match: {date: "2019-07"}},
        {$group: {_id: "$domain", total : {$sum :1}}},
        {$project: {_id: 0,key:"$_id", total:1}},
        {$sort: {"total" : -1}}
    ])
```

- Páginas en las que se han encontrado metadatos fuera de un objeto:

```
db.pages.find({"items.name":"none"})
```

- Cantidad de metadatos encontrados en el elemento head:

```
db.pages.aggregate([
    {$match: {date: "2019-07"}},
    {$unwind: '$items'},
    {$match: {$or: [{"items.tag": {$in: ["head", "html"]}},{"items.name":"none"}]}},
    {$unwind: '$items.properties'},
    {$match: {"items.properties.section" : {$not: { $regex: "body" }}}},
   ]).count()
```

- Cantidad de metadatos encontrados dentro del elemento body:

```
db.pages.aggregate([
    {$match: {date: "2019-07"}},
    {$unwind: '$items'},
    {$match: {$or: [{"items.tag": {$nin: ["head"]}},{"items.name":"none"}]}},
    {$unwind: '$items.properties'},
    {$match: {"items.properties.section" : {$not: { $regex: "head" }}}},
   ]).count()
```

PRESUPUESTO

En este apartado se va a plantear un presupuesto orientativo para el desarrollo, mantenimiento y uso de esta herramienta.

El presupuesto se va a calcular para el desarrollo de la herramienta desde cero y se va a tomar como ejemplo el dataset WARC de CommonCrawl de julio de 2019.

- Desarrollo y mantenimiento:
    - Dos programadores analistas para realizar el estudio de los datasets disponibles, escoger los más apropiados, realizar un diseño del extractor para cada uno y definir el formato de datos y las consultas que permitan obtener la máxima información de los resultados. El coste anual seria de 35.000€ por persona; en total 70.000€.
    - Dos DevOps que se encarguen de la implementación de los extractores de datos, así como del mantenimiento del framework y del desarrollo de una api que pueda ser usada tanto por la interfaz web como por el público. El precio anual seria de 30.000€ por persona; en total 60.000€.
    - Un diseñador y programador frontend para el desarrollo de una web que permita visualizar los resultados obtenidos y acceso al dataset por parte del público. El coste anual sería de 35.000€.
    - No se considera que haya coste de hardware o software para el desarrollo ya que este proyecto se puede desarrollar y mantener a través de teletrabajo de los distintos participantes que utilicen sus propios dispositivos y que utilicen herramientas software gratuitas.
- Máquinas:
    - Almacenamiento: para el almacenamiento de los datos es necesario un servidor de MongoDB. Por ejemplo el coste mensual de una instancia db.r4.xlarge de Amazon DocumentDB seria de (0.554/hora * 24 *30) aproximadamente 400€ al mes o 4.800€ al año.
    - Alojamiento Web: es necesario un servicio que de acceso a la interfaz web. Un ejemplo sería una instancia de EC2 de tipo m4.xlarge con un coste de 0.18€ a la hora lo que sumaría (0.18*20*30) 130€ al mes o 1.560€ al año.
    - Extraccion de datos: tomando por ejemplo la cantidad de datos existente para el dataset WARC de Common Crawl (56000 archivos) y tomando como tiempo medio estimado de procesamiento 6 minutos por archivo (si el procesamiento no se realiza en AWS el tiempo de procesamiento puede aumentar drásticamente en función del ancho de banda) el coste que supondría extraer un dataset entero utilizando seria de:

      56000 archivos * (6 minutos/archivo) / (60 minutos/hora) = 5600 horas de procesamiento.

      Utilizando 8 instancias de tipo c5.2xlarge (8 CPUs) paralelamente con un coste máximo de 0.0815 a la hora el total sería:

      5600 horas / (8 hilos/instancia) = 700 horas de instancias ejecutándose

700 horas * 0.0815 = 57 € por extracción

700 horas / 8 instancias = 87.5 horas en total

Si se desean extraer todos los datasets que Common Crawl publica al año el total ascendería a 684€ al año.

En base a estos datos, el coste del primer año para desarrollar este proyecto sería de 172.044 €.

BIBLIOGRAFÍA

Common Crawl (2019), https://commoncrawl.org/

[1] Information and documentation – The WARC File Format (2008), http://bibnum.bnf.fr/WARC/WARC_ISO_28500_version1_latestdraft.pdf

[2] Schema.org (2019), https://schema.org/

[3] WebSchemas/Accessibility – W3C Wiki (2019), https://www.w3.org/wiki/WebSchemas/Accessibility

[4] HTML Microdata (2019), https://www.w3.org/TR/microdata/

[5] RDFa Core 1.1 (2019), https://www.w3.org/TR/rdfa-core/

[6] JSON-LD 1.1 (2019), https://www.w3.org/2018/jsonld-cg-reports/json-ld/

[7] Web Data Commons (2019), http://webdatacommons.org/

[8] Amazon EC2 (2019), https://aws.amazon.com/ec2/

[9] AWS Alamacenamiento de datos seguro en la nube (2019), https://aws.amazon.com/es/s3

[10] Amazon Simple Queue Service (2019), https://aws.amazon.com/es/sqs/

[11] MongoDB (2019), https://www.mongodb.com/

[12] Node.js (2019) https://nodejs.org/en/

[13] Angular (2019) https://angular.io/

APÉNDICE A. GLOSARIO

A

- **Amazon Web Services**: colección de servicios de computación en la nube ofrecidos a través de internet por Amazon.com.
- **AWS**: Vease "Amazon Web Services"

E

- **EC2**: (Elastic Compute Cloud) Servicio de AWS que proporciona capacidad informática en la nube a través del alquiler de máquinas virtuales.

H

- **HTML**: (Hypertext Markup Language) Lenfuaje de marcado estándar para documentos diseñados para ser mostrados en navegadores web.
- **HTTP**: (Hypertext Transfer Protocol) Protocolo de aplicación para sistemas informáticos distribuidos y colaborativos.

S

- **S3**: (Simpe Storage Service) Servicio de AWS que proporciona almacenamiento seguro en la nube a través de una interfaz de servicio web.
- **SQS**: (Simple Queue Service) Servicio de AWS que proporciona un servicio de mensajería distribuido a través de un sistema de colas.

W

- **WARC**: (Web ARChive) Formato de archivo que especifica un método de combinar múltiples recursos digitales en un solo archivo.
- **WAT**: (Web Archive Transformation) Especificación que describe una forma estructurada de almacenar metadatos.
- **WET**: Especificación que describe como almacenar texto relacionado con diferentees recursos digitales en un solo archivo.
- **W3C**: (World Wide Web Consortium) Organización que se encarga de producir los estándares que se deben usar en la web.

APÉNDICE B. CÓDIGO

En este apéndice se va a incluir el código relacionado únicamente con la extracción de datos una vez proporcionado el archivo.

1 FileExtractor

```
package amef.processor;


import java.util.regex.Pattern;


public interface FileExtractor {


    String MICRODATA = "Microdata";
    String RDFa = "RDFa";
    String JSON = "JSON-LD";


    String ITEMSCOPE = "itemscope";
    String ITEMTYPE = "itemtype";
    String ITEMPROP = "itemprop";


    String VOCAB = "vocab";
    String TYPEOF = "typeof";
    String PROPERTY = "property";


    String SCRIPT = "script";
    String SCRIPTTYPE = "type";
    String JSONTYPE = "ld+json";
    String CONTEXT = "@context";
    String TYPE = "@type";
    String GRAPH = "@graph";



    String CONTENT = "content";


    String NONE = "none";
```

```
    Pattern SCHEMAURL = Pattern.compile("https?://schema\\.org.*");

    String HEAD = "head";
    String BODY = "body";
    String HTML = "html";
}
```

2 FileProcessor

```
    package amef.processor;

    import java.io.InputStream;

    public interface FileProcessor {
        void process(InputStream fileStream, String inputFileKey) throws Exception;
    }
```

3 SchemaWarcProcessor

```
    package amef.processor;

    import amef.ProcessingNode;
    import org.apache.commons.io.IOUtils;
    import org.apache.log4j.Logger;
    import org.bson.Document;
    import org.jsoup.Jsoup;
    import org.jwat.warc.WarcReader;
    import org.jwat.warc.WarcReaderFactory;

    import java.io.ByteArrayInputStream;
    import java.io.IOException;
    import java.io.InputStream;
    import java.nio.charset.IllegalCharsetNameException;
    import java.nio.charset.UnsupportedCharsetException;
```

```
import java.util.*;



import amef.schema.SchemaItem;
import org.jwat.warc.WarcRecord;

public class SchemaWarcProcessor extends ProcessingNode implements FileProcessor {

   private static Logger log = Logger.getLogger(SchemaWarcProcessor.class);

   @Override
   public void process(InputStream fileStream,
                       String inputFileKey) throws Exception {

      log.info("Extracting data from " + inputFileKey + " ...");
      WarcReader warcReader = WarcReaderFactory.getReaderCompressed(fileStream);

      long pagesTotal = 0;
      long foundTotal = 0;
      long start = System.currentTimeMillis();

      String date = getDate(inputFileKey);

      // read all entries in the ARC file
      RecordWithOffsetsAndURL item;
      item = getNextResponseRecord(warcReader);
      List<Document> result = new ArrayList<>();
      while (item != null) {
         List<SchemaItem> items;
         try {
            org.jsoup.nodes.Document doc;
            try {
               // try parsing with charset detected from doc
               doc = Jsoup.parse(new ByteArrayInputStream(item.bytes),
                     null, "");
```

```
            items =  new SchemaHTMLExtractor(search()).extract(doc);
        } catch (IllegalCharsetNameException
                | UnsupportedCharsetException e) {
            try {
                // didnt work, try parsing with utf-8 as charset
                doc = Jsoup.parse(new ByteArrayInputStream(item.bytes),
                        "UTF-8", "");
                items =  new SchemaHTMLExtractor(search()).extract(doc);
            } catch (IllegalCharsetNameException
                    | UnsupportedCharsetException e2) {
                // didnt work either, no result
                items = new ArrayList<>();
            }
        }
        if (!items.isEmpty()) {
            result.add(getDocument(inputFileKey,items,item.url,date));
            foundTotal++;
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        pagesTotal++;
    }

    // next record with one retry
    item = getNextResponseRecord(warcReader);
}
warcReader.close();

//Store statistics in a document
double duration = (System.currentTimeMillis() - start) / 1000.0;
double rate = (pagesTotal * 1.0) / duration;

Document fileStats = new Document();
fileStats.append("file",inputFileKey);
```

```
    fileStats.append("total", date);
    fileStats.append("ready",pagesTotal);
    fileStats.append("found",foundTotal);
    fileStats.append("duration",duration);
    fileStats.append("rate",rate);

    //Upload results to database
    getOutStorage().store(getOutputFileKey(date,inputFileKey),fileStats,result);

    log.info("Extracted data from " + inputFileKey + " - parsed "
            + pagesTotal + " pages in " + duration + " seconds, " + rate
            + " pages/sec");
}

private Document getDocument(String file, List<SchemaItem> items, String url, String date){
    Document result = new Document();
    result.append("file",file);
    result.append("date",date);
    result.append("domain", getDomain(url));
    result.append("url", url);
    result.append("items", items);
    return result;
}


private String getDate(String fileName){
    int slashIndex = fileName.lastIndexOf('/');
    String date = fileName.substring(slashIndex+9, slashIndex+15);
    return date.substring(0,4).concat("-").concat(date.substring(4));
}

protected String getDomain(String url){
    String domain = url.substring(url.indexOf(':')+3);
    return domain.contains("/") ? domain.substring(0, domain.indexOf('/')) : domain;
}
```

```
private String getOutputFileKey(String date, String inputFileKey){
   int idx = inputFileKey.indexOf(".warc");
   String key = inputFileKey.substring(0, idx) + ".json";
   return date + "/" + key;
}
```

```
private static final String WARC_TARGET_URI = "WARC-Target-URI";

protected static class RecordWithOffsetsAndURL {
   public byte[] bytes;
   public String url;

   public RecordWithOffsetsAndURL(byte[] bytes, String url) {
      super();
      this.bytes = bytes;
      this.url = url;
   }
}

protected RecordWithOffsetsAndURL getNextResponseRecord(WarcReader warcReader)
      throws IOException {
   WarcRecord wr;
   while (true) {
      try {
         wr = warcReader.getNextRecord();
      } catch (IOException e) {
         continue;
      }
      if (wr == null)
         return null;
```

```
            String type = wr.getHeader("WARC-Type").value;
            if (type.equals("response")) {
                byte[] rawContent = IOUtils.toByteArray(wr.getPayloadContent());
                String url = wr.getHeader(WARC_TARGET_URI).value;
                return new RecordWithOffsetsAndURL(rawContent, url);
            }
        }
    }
}
```

4 SchemaHTMLExtractor

```
package amef.processor;

import amef.schema.SchemaItem;
import amef.schema.SchemaProperty;
import com.google.gson.*;
import org.jsoup.nodes.Element;
import org.jsoup.nodes.Node;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.Stack;

/**
 *  This class
 */

public class SchemaHTMLExtractor implements FileExtractor{


    private List<SchemaItem> items = new ArrayList<>();
```

```
private Stack<SchemaItem> microdataStack = new Stack<>();
private Stack<SchemaItem> rdfaStack = new Stack<>();

private Stack<String> vocabStack = new Stack<>();

private String section = "html";

private Map<String,List<String>> search;

public SchemaHTMLExtractor(Map<String,List<String>> search){
   this.search = search;
}

public List<SchemaItem> extract(Node root){
   traverse(root);
   if(!microdataStack.isEmpty()) System.out.println("microdataStack not empty");
   if(!rdfaStack.isEmpty()) System.out.println("rdfaStack not empty");
   if(!vocabStack.isEmpty()) System.out.println("vocabStack not empty");
   return items;
}

/**
 *  Performs a depth-first search of the graph
 */
public void traverse(Node root) {
   Node node = root;
   int depth = 0;

   while (node != null) {
      head(node);
      if ( node.childNodeSize() > 0) {
         node = node.childNode(0);
         depth++;
      } else {
         while (node.nextSibling() == null && depth > 0) {
```

```
                tail(node);
                node = node.parentNode();
                depth--;
            }
            tail(node);
            if (node == root)
                break;
            node = node.nextSibling();
        }
    }
}

/**
 *  Called the first time the node is found
 * @param node
 */
private void head(Node node){
    if( node instanceof Element){
        Element elem = (Element) node;
        if (elem.tagName().equals(HEAD) || elem.tagName().equals(BODY)) {
            section = elem.tagName();
        }

        headMicrodata(elem);
        headRDFa(elem);
        headJsonLD(elem);

        if(elem.tagName().equals(HTML)){
            if (microdataStack.isEmpty()) microdataStack.push(new SchemaItem(MICRODATA,NONE,NONE));
            if (rdfaStack.isEmpty()) rdfaStack.push(new SchemaItem(RDFa,NONE,NONE));
        }
    }
}

private void headMicrodata(Element elem){
```

```
        if(elem.hasAttr(ITEMSCOPE)){
            String itemtype = elem.attr(ITEMTYPE);
            if ((SCHEMAURL).matcher(itemtype).matches())
                microdataStack.push(new SchemaItem(MICRODATA,parseItemtype(itemtype),elem.tagName()));
        }
        if (elem.hasAttr(ITEMPROP)){
            String itemprop = elem.attr(ITEMPROP);
            if(search.keySet().contains(itemprop)){
                SchemaProperty prop = new SchemaProperty(itemprop, elem.attr(CONTENT));
                if(microdataStack.size()==1) prop = prop.section(section);
                microdataStack.peek().addProperty(prop);
            }
        }
    }

    private void headRDFa(Element elem){
        if (elem.hasAttr(VOCAB)){
            vocabStack.push(elem.attr(VOCAB));
        }
        if(!vocabStack.isEmpty() && (SCHEMAURL).matcher(vocabStack.peek()).matches() && elem.hasAttr(TYPEOF)) {
            rdfaStack.push(new SchemaItem(RDFa,elem.attr(TYPEOF),elem.tagName()));
        }
        if (elem.hasAttr(PROPERTY)){
            String property = elem.attr(PROPERTY);
            if(search.keySet().contains(property)){
                SchemaProperty prop = new SchemaProperty(property, elem.attr(CONTENT));
                if(rdfaStack.size()==1) prop = prop.section(section);
                rdfaStack.peek().addProperty(prop);
            }
        }

    }

    private void headJsonLD(Element elem){
```

```
        if(elem.tagName().equals(SCRIPT) && elem.attr(SCRIPTTYPE).contains(JSONTYPE)){
            try{
                JsonElement jsonElement = new JsonParser().parse(elem.text());
                if(jsonElement.isJsonObject()){
                    JsonObject object = jsonElement.getAsJsonObject();
                    if(object.has(CONTEXT) &&
(SCHEMAURL).matcher(object.get(CONTEXT).getAsString()).matches()){
                        extractJsonLD(object);
                    }
                }
            }catch (JsonSyntaxException mje){
                // void
            }
        }
    }

    private void extractJsonLD(JsonObject object){
        if(object.has(GRAPH) && object.get(GRAPH).isJsonArray()){
            JsonArray graph = object.get(GRAPH).getAsJsonArray();
            for (JsonElement graphElem: graph) {
                if(graphElem.isJsonObject()) extractJsonLD(graphElem.getAsJsonObject());
            }
        }
        if(object.has(TYPE)){
            SchemaItem item = new SchemaItem(JSON,object.get(TYPE).getAsString(),SCRIPT);
            for(String metaProperty : search.keySet()){
                if(object.has(metaProperty)){
                    JsonElement metaElem = object.get(metaProperty);
                    if(metaElem.isJsonArray()){
                        JsonArray metaArray = metaElem.getAsJsonArray();
                        for(JsonElement propElem : metaArray){
                            if(propElem.isJsonPrimitive()){
                                String prop = propElem.getAsString();
                                item.addProperty(new SchemaProperty(metaProperty,prop));
                            }
```

```
                    }
                }else if(metaElem.isJsonPrimitive()){
                    String prop = metaElem.getAsString();
                    item.addProperty(new SchemaProperty(metaProperty,prop));
                }
            }
        }
        if(!item.getProperties().isEmpty()) items.add(item);
    }
}

/**
 *  Called when the node is exited
 * @param node
 */
private void tail(Node node) {
    if (node instanceof Element) {
        Element elem = (Element) node;
        if ((elem.hasAttr(ITEMSCOPE) && (SCHEMAURL).matcher(elem.attr(ITEMTYPE)).matches()) ||
elem.tagName().equals(HTML)) {
            SchemaItem microdataItem = microdataStack.pop();
            if(!microdataItem.getProperties().isEmpty()) items.add(microdataItem);
        }
        if ((!vocabStack.isEmpty() && (SCHEMAURL).matcher(vocabStack.peek()).matches() &&
elem.hasAttr(TYPEOF)) || elem.tagName().equals(HTML)) {
            SchemaItem rdfaItem = rdfaStack.pop();
            if(!rdfaItem.getProperties().isEmpty()) items.add(rdfaItem);
        }
        if (elem.hasAttr(VOCAB)){
            vocabStack.pop();
        }
    }

}

private String parseItemtype(String url){
```

```
            return url.substring(url.lastIndexOf('/')+1);
        }
    }
```

5 SchemaWDCProcessor

```
package amef.processor;

import amef.ProcessingNode;
import amef.schema.SchemaItem;
import amef.schema.SchemaProperty;
import org.apache.log4j.Logger;
import org.bson.Document;

import java.io.BufferedReader;
import java.io.EOFException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.util.zip.GZIPInputStream;


public class SchemaWDCProcessor extends ProcessingNode implements FileProcessor, FileExtractor {

    private static Logger log = Logger.getLogger(SchemaWDCProcessor.class);

    @Override
    public void process(InputStream fileStream, String inputFileKey) throws Exception {

        log.info("Extracting data from " + inputFileKey + " ...");

        String date = "";
```

```
Matcher dateMatcher = Pattern.compile("\\d{4}-\\d{2}").matcher(inputFileKey);
if(dateMatcher.find()){
   date = dateMatcher.group();
}

long pagesTotal = 0;
long foundTotal = 0;
long start = System.currentTimeMillis();

List<Document> result = new ArrayList<>();
String format = "unknown";
if(inputFileKey.contains("html-microdata")){
   format = MICRODATA;
}else if (inputFileKey.contains("html-rdfa")){
   format = RDFa;
}else if (inputFileKey.contains("html-embedded-jsonld")){
   format = JSON;
}

try (BufferedReader reader = new BufferedReader(new InputStreamReader(new GZIPInputStream(fileStream)))) {
   String currentUrl = "";
   Map<String,SchemaItem> items = new HashMap<>();
   while (reader.ready()){
      String line;
      try{
         line = reader.readLine();
      } catch (EOFException e){
         break;
      }
      StringTokenizer st = new StringTokenizer(line);
      String ref, type, value = "", url;
      try{
         ref = removeMarks(st.nextToken());
         type = removeMarks(st.nextToken());
         while (st.countTokens() > 2){
```

```
            value+=st.nextToken();
        }
        value = removeMarks(removeQuotes(value));
        url = removeMarks(st.nextToken());
    } catch (NullPointerException e){
        continue;
    }
    if(!url.equals(currentUrl)){
        List<SchemaItem> foundItems = new ArrayList<>();
        for(SchemaItem item : items.values()){
            if(!item.getProperties().isEmpty()) foundItems.add(item);
        }
        if(!foundItems.isEmpty()){
            result.add(getDocument(inputFileKey,foundItems,currentUrl,date));
            foundTotal++;
        }
        pagesTotal++;
        currentUrl = url;
        items.clear();
    }
    if(type.equals("http://www.w3.org/1999/02/22-rdf-syntax-ns#type")){
        if(SCHEMAURL.matcher(value).matches())
            items.put(ref,new SchemaItem(format,value.substring(value.lastIndexOf('/')+1)));
    } else {
        if (!items.containsKey(ref)){
            items.put(ref,new SchemaItem(format,"none"));
        }
        if(SCHEMAURL.matcher(type).matches()){
            String prop = type.substring(type.lastIndexOf('/')+1);
            if(search().keySet().contains(prop)){
                if(value.contains("@")) value = value.substring(0,value.indexOf('@'));
                StringTokenizer values = new StringTokenizer(value,",");
                while (values.hasMoreTokens()){
                    String val = values.nextToken();
                    if(search().get(prop) == null) items.get(ref).addProperty(new SchemaProperty(prop,val));
```

```
                    else {
                        for(String acceptedValue : search().get(prop)){
                            if(val.startsWith(acceptedValue)) items.get(ref).addProperty(new SchemaProperty(prop,val));
                        }
                    }
                }
            }
        }
    }

    //Store statistics in a document
    double duration = (System.currentTimeMillis() - start) / 1000.0;
    double rate = (pagesTotal * 1.0) / duration;

    Document fileStats = new Document();
    fileStats.append("file",inputFileKey);
    fileStats.append("date", date);
    fileStats.append("total",pagesTotal);
    fileStats.append("found",foundTotal);
    fileStats.append("duration",duration);
    fileStats.append("rate",rate);

    if(!result.isEmpty()) getOutStorage().store(inputFileKey, fileStats, result);
  }
}

private Document getDocument(String file, List<SchemaItem> items, String url, String date){
  Document result = new Document();
  result.append("file",file);
  result.append("date",date);
  result.append("domain", getDomain(url));
  result.append("url", url);
  result.append("items", items);
  return result;
```

```
    }

    protected String getDomain(String url){
        String domain = url.substring(url.indexOf(':')+3);
        return domain.contains("/") ? domain.substring(0, domain.indexOf('/')) : domain;
    }

    private String removeMarks(String s){
        return s.replace("<","").replace(">","");
    }

    private String removeQuotes(String s){
        return s.replace("\"","");
    }
}
```