UNIVERSIDAD EAFIT

Proyecto Final
(Segunda Parte)

DANIEL MÚNERA SÁNCHEZ

JAVIER QUINTERO GONZÁLEZ

TÓPICOS ESPECIALES EN TELEMÁTICA

SEGUNDA ENTREGA PROYECTO FINAL

TÓPICOS ESPECIALES EN TELEMÁTICA

Presentado por:
DANIEL MÚNERA SÁNCHEZ
JAVIER QUINTERO GONZÁLEZ

Presentado a:
EDWIN MONTOYA MÚNERA
PROFESOR

UNIVERSIDAD EAFIT
MEDELLÍN

2012

1. DESCRIPCIÓN DE LA APLICACIÓN O DEL SERVICIO A IMPLEMENTAR.

Modelado y diseño global de la aplicación

Nuestra idea es realizar un tipo de araña de búsqueda más específica, conocida también como Topical Crawler, a la cual le daremos una serie de sitios web de multiplex que operan en el país, en los cuales consultaremos información relevante sobre las películas disponibles, sus carteleras y lugares para verlas dependiendo de la ciudad.

El objetivo principal que tenemos es facilitarle a los usuarios la búsqueda de películas disponibles y horarios, creando un lugar donde se centralice la información, evitando tener que entrar a 3 o más sitios para ver que peliculas estan disponibles.

Pensando en implementación, consideramos dos estrategias:

- **Implementar un Crawler utilizando librerías como Anemone:** Esta estrategia fue muy llamativa y nos llevó a investigar sobre Crawler y Topical Crawlers, pero llegamos a la conclusión de que teniendo las páginas sobre multiplex de la ciudad identificadas (que utilizamos como seeds de búsqueda), era más eficiente realizar búsquedas estáticas y preocuparnos por encontrar la manera de realizar el parsing de la url, para obtener la información requerida, ya que encontramos algunas dificultades para realizarlo.

- **Implementar una búsqueda más específica, utilizando principios de la búsqueda por medio de Topical crawlers:** Llegamos a esta decisión pensando en el performance y haciendo un balance entre beneficio y costo de implementación. También nos basamos en un ejemplo real de una empresa Brasileña que realiza algo similar a nuestro trabajo y utilizando esta estrategia. (http://sieve.com.br/)

Explicación de la estrategia a utilizar:

Haciendo una análisis de la información que tenemos que recuperar, encontramos los siguientes datos:

1. Ciudades donde están los multiplex
2. Lugares
3. Películas
4. Horarios

Características de los sitios web de los multiplex que vamos a utilizar para obtener la información:

Para esta práctica queremos empezar en 2 sitios:

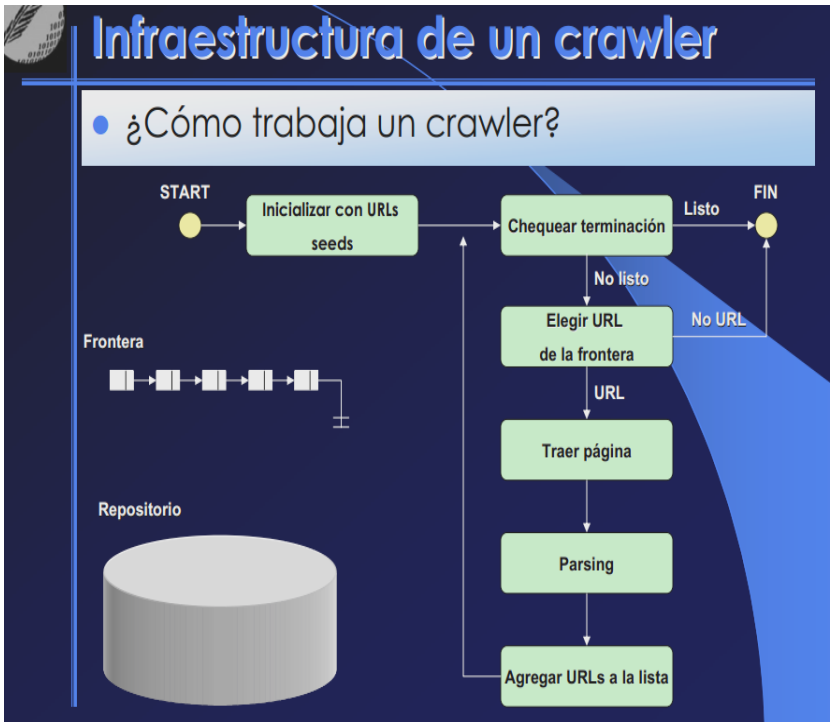
- Cine Colombia
- Procinal

Estos sitios web no tienen ningún web services el cual pudiéramos aprovechar para obtener la información requerida, por tal motivo pensamos en utilizar estos

sitios web como semillas de una búsqueda y desde ahí identificar en qué parte del sitio web se encontraba la información requerida. El proceso consiste:

1. Tener identificadas las URL's relevantes para nosotros.
2. Hacer un request pidiendo el HTML de la página web.
3. Hacer un parsing y obtener la información necesaria.

Con el siguiente modelo explicaremos qué conceptos de Topical Crawler utilizamos para crear nuestra estrategia:



Aquí se muestra el proceso general de un crawler, inicia con unos seeds, desde los cuales empieza a navegar y a buscar enlaces dentro del sitio y mientras sea necesario sigue buscando y va guardando los sitios que son relevantes para la búsqueda. Luego de ese procedimiento se procede al parsing para guardar la información relevante.

En esencia ese es el mismo procedimiento que nosotros vamos a seguir para obtener la información, solo que con un conjunto de seeds muy reducidos y con Parsers específicos para cada seed utilizada.

Manejo de la información y segmentación de la información a buscar:

Luego realizar un análisis del problema identificamos dos tipos de información a buscar:

1. **Información estática:** Es la información que es necesaria para realizar la búsqueda con éxito que no está sujeta a cambios en el corto plazo. Estos datos son:
   - Ciudades
   - Lugares

   Es un trabajo de mucho tiempo poder montar un multiplex en una ciudad nueva, por ese motivo creemos que no es necesario recolectar esta

información cada vez que se realice una búsqueda. Pensamos darle la opción a un administrador del sitio de actualizar esta información, o crear un proceso que se ejecute cada mes o una ventana de tiempo similar para actualizar esa información

2. **Información dinámica:** Es la información que está cambiando constantemente y necesita ser obtenida cada vez que se realice una búsqueda. Estos datos son:
    - Películas disponibles
    - Horarios

Teniendo ya esta clasificación de la información decidimos, crear una base de datos para guardar la información estática y ayudar a volver la búsqueda más rápida, ya que son dos ciclos de trabajo idénticos al del topical crawler que eliminamos. Además la carga de la información general para poder iniciar la búsqueda también se cargará más rápido, ya que las ciudades y los multiplex se cargaran localmente en los selectores al iniciar la aplicación.

Detalles a tener en cuenta:

Algunos detalles que son críticos para nosotros son:

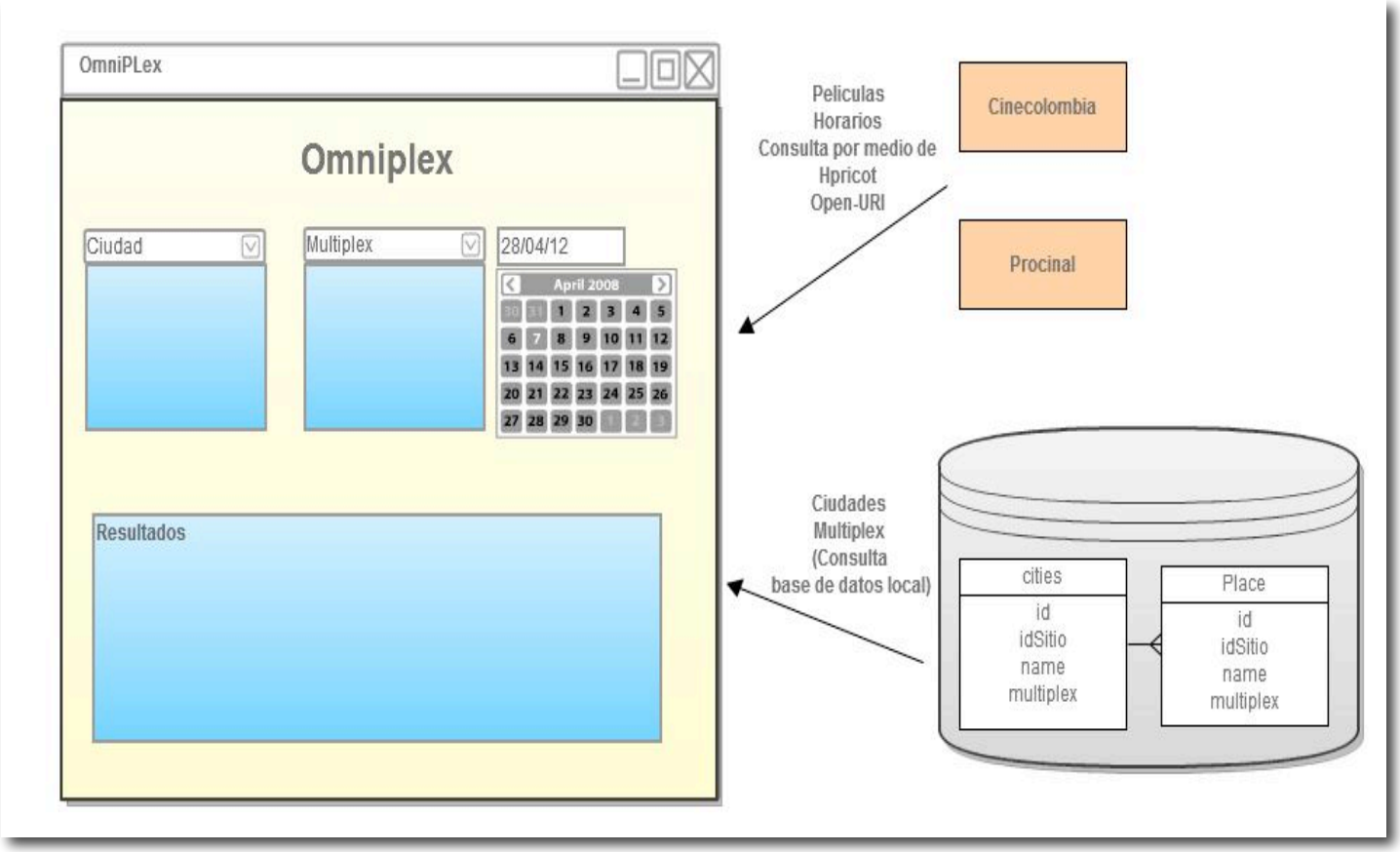
- **El manejo e integridad de la información:** Como sabemos, cada página tiene su forma de presentar la información, por ejemplo, el nombre de las ciudades puede estar en encoding UTF-8 u otro, o puede tener ciertas diferencias entre mayúsculas y minúsculas. Por lo que tenemos que buscar algoritmos para encontrar la distancia entre strings e identificar cuando dos ciudades son diferentes y cuando no. Y así unificar el listado de ciudades de todos los multiplex que se van agregando al sistema.

- El diseño de los parsers puede ser complicado, debido a la estructura de algunos sitios que en muchas ocasiones tienen contenido cargado dinámicamente con JavaScript o contenido en flash. En algunos casos las páginas son una combinación de sitios web y por tal las carteleras de las diferentes ciudades tienen caminos muy diferentes para ser encontradas.

- El tiempo de búsqueda tiene que ser corto y efectivo para que la aplicación pueda ser exitosa.

modelo y diseño global

Teniendo un poco más claros los detalles y el estado del arte de nuestro problema, llegamos al siguiente modelo para mostrar la lógica de interacción.



2. MARCO DE REFERENCIA Y ESTADO DEL ARTE DEL MODELO EMERGENTE

La enorme cantidad de información disponible en la web ha hecho necesario el desarrollo de recursos que permitan encontrar de manera eficiente y precisa la información requerida por el usuario. Dentro de las diversas técnicas desarrolladas crawling es capaz de rastrear determinados tópicos de manera rápida sin tener que explorar todas las páginas web.

Mirando la importancia de este tipo de herramientas decidimos desarrollar una **aplicación web basada en los conceptos de crawling**, especificamente del concepto de topical crawling. El sistema solo consiste en una aplicación web, que tiene la capacidad recopilar información distribuida en la web y exponerla al usuario para su facil acceso. A continuación exponemos una explicación de los conceptos en los cuales nos basamos para sustentar nuestro trabajo.

Topic -specific web crawler han sido desarrollados con el objetivo de recopilar páginas de internet eligiendo sólo las relacionadas con un tema en particular. Algunos crawler desarrollados son:

- **WTMS:** Web Topic Management System. Para recolectar información relacionada con un tema en particular el usuario tiene que especificar algunas urls relevantes que se convertirán en la base para las búsquedas. El WTMS descarga las urls base y crea un documento vector representativo (RDV) basado en la frecuencia de ocurrencia de las palabras claves en esas páginas. A continuación se descargan las páginas que son referenciadas por las urls semilla y se calcula su similitud con el RDV que ha sido definido previamente y se pasa un rango establecido se añaden a una cola. Se continúa de esta manera hasta que la cola está llena o se establece un límite. Después de rastreo, la colección se compone de las urls semilla, así como todas las páginas similares a estas urls de semillas que tienen rutas hacia o desde las semillas.

  Esta colección es una buena fuente de información disponible en la web para el tema especificado por el usuario. El rastreador tiene una lista de direcciones URL de parada para evitar la descarga de algunas páginas populares (como Yahoo y las páginas de Netscape), como parte de la colección.

- **Shark-Search:** se basa en el algoritmo Fish-Search en el cual la clave principal es la siguiente: se toma como entrada una URL semilla y una cola de búsqueda y dinámicamente se crea una lista de prioridades (inicializada con las URL semilla) de las siguientes URLs (llamadas nodos) ser exploradas. En cada paso el primer nodo se extrae de la lista y es procesado. Cada texto es analizado y evaluado de si es relevante o no y se decide si se continúa explorando en esta dirección o no. Cuando un documento procedente de una URL es recuperado se analiza en busca de Links. Los nodos procedentes de este documento son llamados hijos y le es asignado un valor de profundidad. Si el padre es relevante, la profundidad del hijo es un conjunto de valores predefinidos, de otra manera, la profundidad de los hijos es un conjunto que es un grado menor que el del padre. Cuando la profundidad llega a cero, la dirección se elimina y ninguno de los hijos es insertado en la lista.

  Algunas mejoras son hechas a este algoritmo y ahora se utiliza una puntuación "difusa" de relevancia, dando al hijo una puntuación heredada, por lo que prefieren los hijos de un nodo que tiene un mejor resultado. Una mejora más significativa consiste en refinar el cálculo de la puntuación potencial de los hijos no sólo mediante la propagación de las puntuaciones más relevantes de los ancestros de relevancia en la jerarquía, sino también haciendo uso de la meta-información contenida en los enlaces a los documentos.

- **Get** as Input parameters, the initial node, the width *(width)*, depth *(D)* and size *(S)* of the desired graph to be explored, the time limit, and a search query
- **Set** the depth of the initial node as *depth* = *D*, and **Insert** it into an empty list
- **While** the list is not empty, and the number of processed nodes is less than S, and the time limit is not reached
    1. **Pop** the first node from the list and make it the current node
    2. **Compute** the relevance of the current node
    3. **If** *depth* > *0*:
        1. **If** *current_node* is irrelevant
           Then
            - **For** each child, *child_node*, of the first *width* children of *current_node*
                - **Set** *potential_score(child_node)* = *0.5*
            - **For** each child, *child_node*, of the rest of the children of *current_node*
                - **Set** *potential_score(child_node)* = *0*

           Else
            - **For** each child, *child_node*, of the first *(a * width)* children of *current_node*
              (where *a* is a pre-defined constant typically set to *1.5*)
                - **Set** *potential_score(child_node)* = *1*
            - **For** each child, *child_node*, of the rest of the children of *current_node*
                - **Set** *potential_score(child_node)* = *0*
        2. **For** each child, *child_node*, of *current_node*,
            - **If** *child_node* already exists in the priority list,
              Then
                1. **Compute** the maximum between the existing score in the list to the newly computed potential score
                2. **Replace** the existing score in the list by that maximum
                3. **Move** *child_node* to its correct location in the sorted list if necessary

              Else Insert *child_node* at its right location in the sorted list according to its *potential_score* value
        3. **For** each child, *child_node*, of *current_node*,
            - **Compute** its depth, *depth(child_node)*, as follows:
                1. **If** *current_node* is relevant,
                   Then Set *depth(child_node)* = *D*
                   Else *depth(child_node)* = *depth(current_node) - 1*
                2. **If** *child_node* already exists in the priority list
                   Then
                    1. **Compute** the maximum between the existing depth in the list to the newly computed depth
                    2. **Replace** the existing depth in the list by that maximum.
- **EndWhile**

Figure 1: The Fish-Search Algorithm

- **InfoSpider:** busca online información relevante para el usuario, tomando decisiones autónomas sobre qué enlaces seguir. Una parte importante del sistema es que el usuario puede regenerar la importancia de un documento visitado por InfoSpiders, esto ocurre de forma asíncrona con respecto a la búsqueda en línea, y alterar el comportamiento subsecuente de los agentes, cambiando el ambiente de búsqueda. Es decir, el usuario obra recíprocamente con el ambiente para predisponer el proceso de la búsqueda. El usuario proporciona inicialmente una lista de palabras claves y una lista de los puntos de partida, en forma de un archivo de bookmark. Cada agente es asignado a un enlace sobre el que actúa según su representación.

Una fase de gran importancia dentro del proceso realizado por el crawler es la fase de conocimiento en la cual se destaca la capacidad de aprendizaje del rastreador el cual necesitará como entrada por parte del usuario algunas

palabras claves, o dependiendo del tipo de crawler desarrollado, las URLs base. Este proceso consta de tres fases:

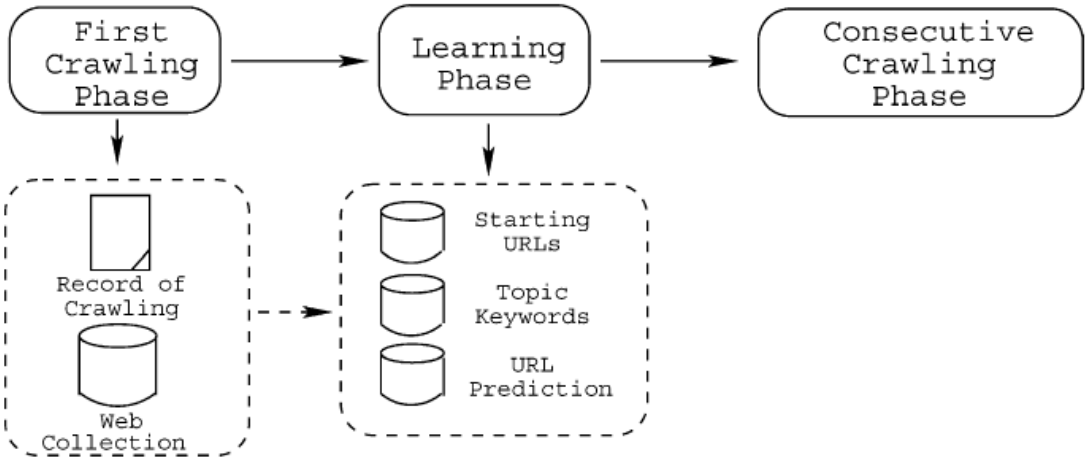


Fig. 1. The diagram of learnable crawling process comprises of three phases: first crawling, learning and consecutive crawling.

- **First Crawling Phase:** durante esta fase el rastreador sólo tiene palabras clave que describen un tema de interés suministrado por el usuario. El rastreador a continuación envia las palabras claves a un motor de búsqueda para construir el conjunto candidato de direcciones de partida, esta es una opción, en la otra el usuario da como punto de partida algunas direcciones URL. Los resultados de esta fase son una colección de páginas web y un rastreo de registros. El rastreador utiliza estos resultados para aprender y construir bases de conocimiento en la fase de aprendizaje.

- **Learning Phase:** es una fase en el que el rastreador intenta aprender cómo recolectar de la mejor manera la información teniendo como base las experiencias previas. La información recolectada será categorizada y clasificada según los estándares propios y se almacenarán los que tengan la mejor puntuación.

- **Consecutive crawling phase:** finalmente el rastreador recopila páginas web mediante el uso de las bases de conocimiento creadas que servirá como base para nuevas búsquedas.

Estructura de web crawler

La estructura de un Web Crawler es la siguiente:

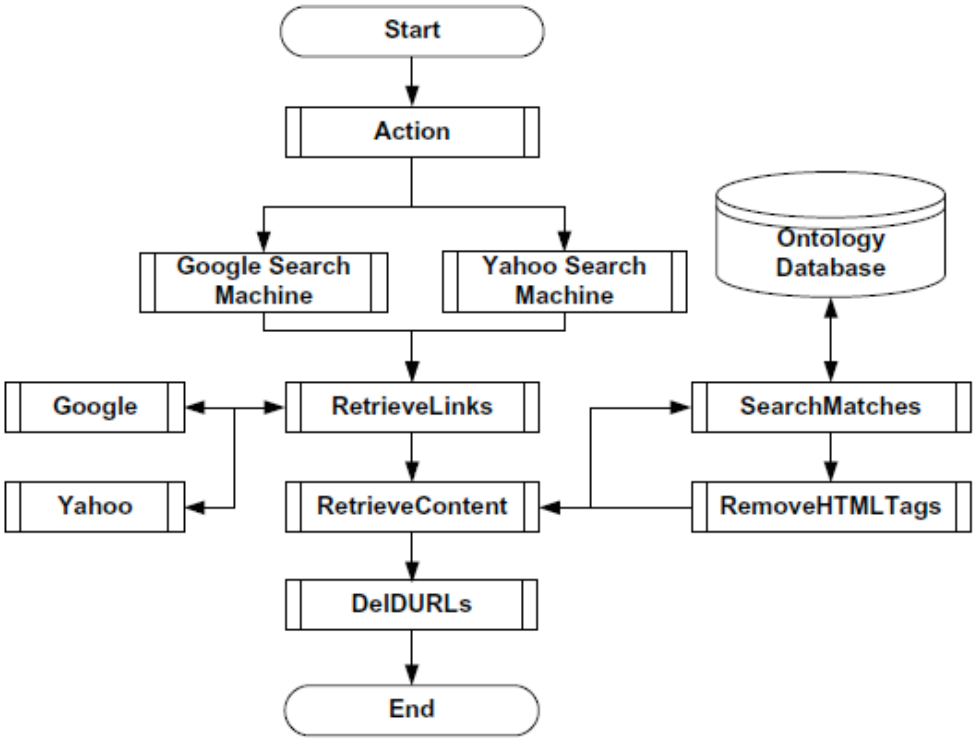


Fig. 4. System structure of Web Crawler.

- **Action:** transforma una consulta interna en una URI y es embedido dentro de una consulta de Google o Yahoo según sea el caso.

- **Google/Yahoo Search Machine:** declara un objeto URL y agrega la URL de Google en la consulta del código URI y entonces se utiliza un bucle interativo para leer su contenido línea por línea. Por último, muestra el contenido como archivo de texto para el análisis final.

- **Retrieve Links:** usa expresiones regulares para buscar la información que ha sido descargada de la URL. REtorna los hipervínculos y la salida de ellos en un archivo de texto para proveer al sistema para nuevos procesamientos.

- **Retrieve Content:** utiliza el archivo de enlace para leer su contenido línea por línea

- **Search Matches:** apoyo interno a la función anterior para saber si la página anterior está dentro del rango que nos interesa consultar. De ser así se añade a la base de datos en la cual estamos guardando la información relevante para nuestra consulta.

- **Remove HTML Tags:** función de apoyo a los dos anteriores y se encarga de eliminar las etiquetas Html de las páginas que han sido seleccionadas.

- **DelDURLs:** continúa con la comparación de las direcciones obtenidas por los motores de búsqueda para evitar la duplicación de páginas.

En términos generales está es la forma en la que opera un Web Crawler, se obtienen páginas de inicio sobre las cuáles se operará, dichas páginas pueden ser asignadas de manera estática (como es nuestro caso) o se consiguen a través de motores de búsqueda como Google o Yahoo. Posteriormente se procede a analizar la información que se ha adquirido, es aquí donde es importante el trabajo del algoritmo de búsqueda que se encargará de clasificar el contenido y mirar si se ajustan a los patrones establecidos (aquí es donde pueden ser utilizados algoritmos como los vistos anteriormente). Finalmente las páginas que han clasificado son añadidas a una base de datos que servirá de base para futuras búsqueda.

3. Selección y documentación del software para el modelo emergente

Para instalar Ruby la mejor alternativa es RVM (Ruby Version Manager): **https://rvm.io//**

Para la realización de la aplicación utilizaremos el lenguaje de programación Ruby, el framework para desarrollo de aplicaciones web Rails, y las siguiente lista de librerías:

- Net/HTTP (**http://ruby-doc.org/stdlib-1.9.3/libdoc/net/http/rdoc/Net/HTTP.html**)
- Hpricot (**https://github.com/hpricot/hpricot**)

- JSON(http://developer.yahoo.com/ruby/ruby-json.html)
- open-uri(http://juretta.com/log/2006/08/13/ruby_net_http_and_open-uri/)
- omniauth(https://github.com/intridea/omniauth)

También existe la posibilidad de utilizar la librería anemone, la cual se utiliza para implementar crawlers con ruby. Todo depende de las decisiones de implementación que tomemos en el camino.

A continuación mostraremos las pruebas que hemos realizado con las librerías que vamos a utilizar:

- Ejemplo para búsqueda de datos utilizando las librerías listadas anteriormente:

Código:

multiplexParser.rb — Ruby

startParser.rb | multiplexParser.rb | anemone.rb

```ruby
require 'open-uri'
require 'rubygems'
require 'hpricot'

class MultiplexParser

  @cities_cinecol
  @movies_cinecol
  @cities_procinal
  def initialize site
    if site =~ /cinecolombia/
      @cities_cinecol = []
      @movies_cinecol = []
      urlcinecol = "http://www.cinecolombia.com/Default.aspx"
      save_cities_cinecol urlcinecol
      movies_cinecol urlcinecol
    elsif site =~ /procinal/
      @cities_procinal = []
      urlprocinal = "http://procinal.com/index.php"
      save_cities_procinal urlprocinal
      movies_procinal urlprocinal
    end
  end

  def save_cities_cinecol url
    cinecol = Hpricot(open(url))
    cities = (cinecol/"select.combociudades/option")
    cities.each do |city|
      val = city.inner_html
      unless /Cambiar Ciudad/.match(val) or /----/.match(val)
        @cities_cinecol.push({:id => city.attributes['value'] , :name => val})
      end
    end
    @cities_cinecol.uniq!
     puts "*** Ciudades disponibles en cinecolombia ***"
      for city in  @cities_cinecol
        puts "=>  ID de pelicula : #{city[:id]} , nombre: #{city[:name]}"
      end
  end
```

```ruby
  def movies_cinecol url
     cinecol = Hpricot(open(url))
     movies = ( cinecol/"select.combopeliculas/option")
     movies.each do |m|
       val = m.inner_html
       @movies_cinecol.push(val) unless /Por Pelicula/.match(val) or /----/.match(val)
     end
     @movies_cinecol.uniq!
     puts "**** Peliculas disponibles en cinecolombia ****"
     for movie in  @movies_cinecol
       puts "=> #{movie}"
     end
  end

  def save_cities_procinal url
    procinal = Hpricot(open(url))
    cities = (procinal/"div.filtrogroup:first ul li a")
    cities.each do |city|
      value = city.inner_html
      @cities_procinal.push(value)
    end
    @cities_procinal.uniq!
    puts "ciudades disponibles en procinal son: #{@cities_procinal}"
  end

  def movies_procinal url
    procinal = Hpricot(open(url))
    movies = (procinal/"div#cuadriculaPeliculas ul li a")
    #puts movies
  end

end
```

Resultados de la ejecución

```
MacBook-Pro-dmuneras:PracticaTelemaco danielmunerasanchez$ ruby startParser.rb cinecolombia
*** Ciudades disponibles en cinecolombia ***
=>  ID de pelicula : 2 , nombre: ARMENIA
=>  ID de pelicula : 3 , nombre: BARRANQUILLA
=>  ID de pelicula : 1 , nombre: BOGOT&#193;
=>  ID de pelicula : 4 , nombre: BUCARAMANGA
=>  ID de pelicula : 5 , nombre: CALI
=>  ID de pelicula : 6 , nombre: CARTAGENA
=>  ID de pelicula : 24 , nombre: MANIZALES
=>  ID de pelicula : 7 , nombre: MEDELL&#205;N
=>  ID de pelicula : 8 , nombre: MONTER&#205;A
=>  ID de pelicula : 9 , nombre: PEREIRA
**** Peliculas disponibles en cinecolombia ****
=> &#161;PIRATAS! UNA LOCA AVENTURA
=> AMERICAN PIE, EL REENCUENTRO
=> BATALLA NAVAL
=> COMANDO ESPECIAL
=> CUANDO TE ENCUENTRE
=> EL CONSPIRADOR
=> EL ENVIADO
=> EL NI&#209;O DE LA BICICLETA
=> EL PRECIO DE LA CODICIA
=> EL ROMANCE DEL SIGLO
=> EL SUE&#209;O DE IVAN
=> EL VENGADOR FANTASMA 2 3D
=> ESPEJITO, ESPEJITO
=> GORDO, CALVO Y BAJITO
=> HOMBRES DE NEGRO III 3D
=> LORAX - EN BUSCA DE LA TRUFULA PERDIDA
=> LOS VENGADORES
=> MUJERES AL PODER
=> OPERA ON ICE
=> VOTOS DE AMOR
MacBook-Pro-dmuneras:PracticaTelemaco danielmunerasanchez$
```

Código ejemplo de anemone:

startParser.rb | multiplexParser.rb | anemone.rb

```ruby
require 'anemone'
require 'open-uri'
require 'hpricot'

unless ARGV.length == 1
  STDERR.puts "Usage: #{$0} <url>"
  exit 1
end

$url = ARGV[0]

puts "Pagina a visitar: #{$url}"

def who_is_it site

  if site =~ /cinecolombia/
    puts "Estamos en cine colombia"
  elsif site =~ /cinemark/
    puts "Estamos en cinemark"
  end
end

who_is_it $url

link = 0
Anemone.crawl($url) do |anemone|
 anemone.on_every_page do |page|
      link = link + 1
      puts page.url
  end
end

puts link.to_s
```

Código en ejecución anemone

```
MacBook-Pro-dmuneras:PracticaTelemaco danielmunerasanchez$ LS
anemone.rb              multiplexParser.rb      startParser.rb
MacBook-Pro-dmuneras:PracticaTelemaco danielmunerasanchez$ ruby anemone.rb http://www.cinecolombia.com/Default.aspx
Pagina a visitar: http://www.cinecolombia.com/Default.aspx
Estamos en cine colombia
http://www.cinecolombia.com/Default.aspx
http://www.cinecolombia.com/contact.aspx
http://www.cinecolombia.com/contactPQRS.aspx
http://www.cinecolombia.com/servicios
http://www.cinecolombia.com/servicios/
http://www.cinecolombia.com/food.aspx
http://www.cinecolombia.com/movie.aspx?idciudad=1&idpelicula=814
http://www.cinecolombia.com/movie.aspx?idciudad=1&idpelicula=820
http://www.cinecolombia.com/movie.aspx?idciudad=1&idpelicula=846
http://www.cinecolombia.com/movie.aspx?idciudad=1&idpelicula=812
http://www.cinecolombia.com/movie.aspx?idciudad=1&idpelicula=854
http://www.cinecolombia.com/movie.aspx?idpelicula=815&idciudad=1
http://www.cinecolombia.com/movie.aspx?idciudad=1&idpelicula=826
http://www.cinecolombia.com/movie.aspx?idciudad=1&idpelicula=646
http://www.cinecolombia.com/movie.aspx?idciudad=1&idpelicula=864
```

Bibliografía

A. Rungsawang*, N. A. (7 de Enero de 2004). *Learnable topic-specific web crawler.* Recuperado el 19 de Mayo de 2012, de http://ac.els-cdn.com/S1084804504000086/1-s2.0-S1084804504000086-main.pdf?_tid=0d09e274ee474e734377edf8311a4fce&acdnat=1337445117_7dd90c03ab3150abec317e34c96a8a89

Chang Su, Y. G. (2005). *An Efficient Adaptive Focused Crawler Based on Ontology Learning.* Recuperado el 19 de Mayo de 2012, de http://csdl.computer.org.ezproxy.eafit.edu.co/dl/proceedings/his/2005/2457/00/24570073.pdf

Fatemeh Ahmadi-Abkenari, A. S. (s.f.). *An architecture for a focused trend parallel Web crawler.* Recuperado el 19 de Mayo de 2012, de http://dl.acm.org/citation.cfm?id=2051686

Luján, R. (s.f.). *Crawling the Web.* Recuperado el 19 de Mayo de 2012, de http://cs.uns.edu.ar/~agm/mineriaweb/downloads/Slides/clase06-slides-rocio.pdf

Martínez, F. (s.f.). *InfoSpider.* Recuperado el 19 de Mayo de 2012, de http://trevinca.ei.uvigo.es/~pcuesta/sm/alumnos2002/infoSpiders.pdf

Michael Hersovici, M. J. (s.f.). *The Shark-Search Algorithm.* Recuperado el 2012 de Mayo de 19, de http://www.cs.cmu.edu/~dpelleg/bin/360.html

Mukherjea, S. (s.f.). *WTMS: A System for Collecting and Analyzing Topic-Specific Web Information.* Recuperado el 19 de Mayo de 2012, de http://www9.org/w9cdrom/293/293.html#Muk00

Yang, S.-Y. (2010). *OntoCrawler: A focused crawler with ontology-supported website models.* Recuperado el 19 de Mayo de 2012, de http://ac.els-cdn.com/S0957417410000205/1-s2.0-S0957417410000205-main.pdf?_tid=9135954b26cd1f0d5a3b8153e0084368&acdnat=1337445082_83a0e61583c1bce367b3c71205010057