Abstract

Accessibility to information being the order of the day, it is of utmost importance how easily one can access the information relevant to his needs.

Web has become the warehouse one looks to access information. Due to the sheer volume of data and unauthentic, unwanted information present, the challenge to seek relevant information has become even more arduous.

Here relevance means most appropriate and connected information. The inherent limitations of conventional key-based searching techniques often neglect the underlying semantics of the information. Here we propose a way to preserve semantics of a documents to yield better search results to end user.

Content from the web is crawled at regular basis, this content is provided as output to user based on his query. Which document is relevant to the query given is the core problem for existing search engines.

This document discusses the method that can be used to infer context on the user's query, as well as the document that is talking about the same context.

TABLE OF CONTENTS

LIST OF TABLES:

LIST OF FIGURES:

LIST OF SYMBOLS AND FAMILIARS:

WORDS TO KNOW:

SYMBOL:	**MEANING:**
Content	Text content from webpage
Crawling	Act of recursive visit from one link to another
Master	Managing module
Slave	Worker module
Fireup	Module used to initialize activities in this project
FFC	Fetch From Crawler module
RGEN	Result Generator module
DMGR	Data Manager module
PDU	Data encapsulator and communicator
Rank	Number indicating trustworthiness of a page
Context	Implied situation,circumstances that alter the meaning of words.
NLP	Matural Language Processing module

2.REQUIREMENT ANALYSIS:

2.1 Abstract Requirements

R1	Receive content from content-retrieval system
R2	Identifying the context
R3	Storing the content in easy-to-retrieve way
R4	Processing the query given by the user
R5	Finding relevant documents
R6	Output the links of relevant documents to the user

Functional Requirements and sub-requirements.

R1. Receiving content from content retrieval system.

R1.1 Booting a network of slaves for a master.

R1.1.1. An application `fireup` to boot the slave network.

R1.1.2. Communication with the A2.Master for configurations of its client.

R1.1.3. Booting up the slaves of the A3 system to connect with the A2.slaves.

R1.2 Fetch the scrapped content from crawlers.

R2. Identifying the context.

R2.1. Categorizing the content.

R3. Storing the webcontent in easy-to-retrieve way (Indexing)

R3.1. Indexing the document based on the keywords.

R3.2. Recieving search queries from `RGEN`s and serve the relevant links.

R4. Query Processing

R4.1. Spell check and unwanted symbol omission.

R5. Fetching the results for the query given by the user

R5.1. Categorizing the query and distributing the query to DMGRs.

R5.2. Sort the links retrieved from the DMGRs and presenting to the user.

R6. Output the links of docments.

R6.1. Links of the documents are wrapped and output to UI.

Non-Functional Requirements and sub-requirements.

NR1. A generic communication protocol for use between any module.

NR2. A webserver which creates a Result generation module `RGEN` for every query submitted.

NR3. Maintaining a proxy-to-original link lookup table.

NR4. A probing system which periodically probes the applications and records their status.

R6.1. heartbeat signals is maintained for each module in the system.

2.2 Concrete Requirements

R1.1.1. An application `fireup` to boot the slave network.
Requirements.
A standalone application -

a. which can be installed in all the machines in the system.
b. which boots the A2.master and A3.master.
c. which connects to master specified at the time of initialization.
d. which recieves the requests from the masters to create, kill different processes specified.
e. which allows the user to manage the processes in that machine.

manage: {create, kill, view errors and output}

R1.1.2. Communication with the A2.Master for configurations of its client.

a. Getting the configuration of the A2.Master.
b. A standard format for configuration of the whole system such as JSON. [NF]

R1.1.3. Booting up the slaves of the A3 system to connect with the A2.slaves.

a. Creating the FFCs and DMGRs for each crawler in the A2 system.
b. Connecting the FFCs and DMGRs.

R1.2 Fetching the scrapped web-content from the connected A2.crawler.

a.Efficient retrieval of content by FFC's adhereing a protocol.

R2.2. Categorizing the content.

a. Choosing a language model for categorizing the web-content.

R3.1. Indexing the document based on the keywords.

a. Storing the minimized document.
b. Indexing the above minimized doc for the keywords in it.

R4.1.1. Spell Check

a.Limited spell correction using dictionary

R4.1.2. Selective Omission

a. Omit unwanted symbols,white spaces to form set of keywords

R5.1. Categorizing the query and distributing the query to DMGRs.

a. Categorizing the query using some language model or sentment analyzer.
b. Sending the query to respective `DMGR`s for getting the links.

R5.2. Sort the links retrieved from the DMGRs

a. sorting is decreasing order of ranking

R6.1. Output the sorted results to UI.

a.wrap the whole set of links in PDU and send to UI module.

NR.1

a. A generic protocol in *JSON* and its detailed description.

NR.2

a. A webserver to display the html page and generate appropriate responses.
b.The UI aspect of the page.The template and details about UI is documented here

ARCHITECTURE :

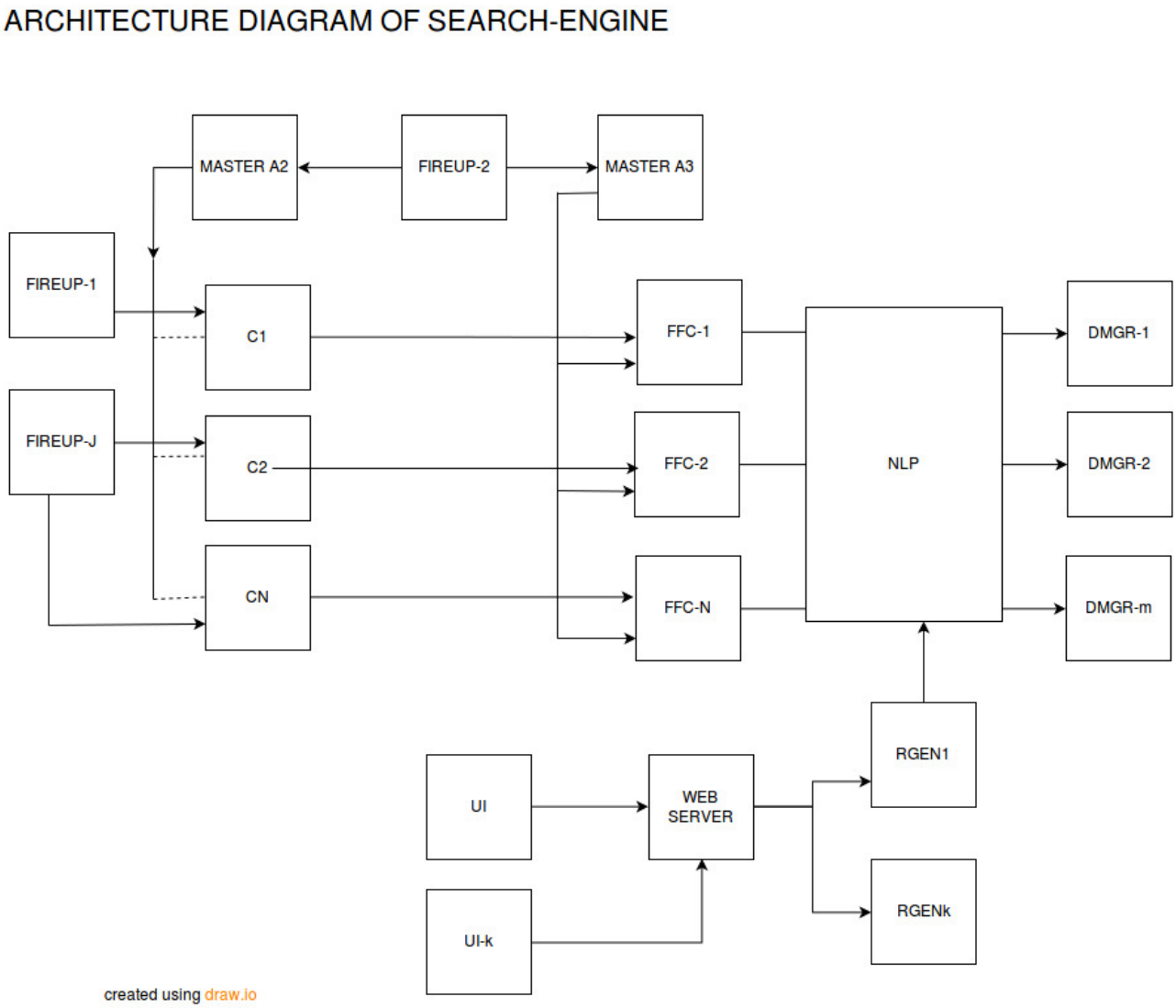

fig.1 Architecture Diagram

DESIGN:

i)STARTUP SEQUENCE OF SLAVES

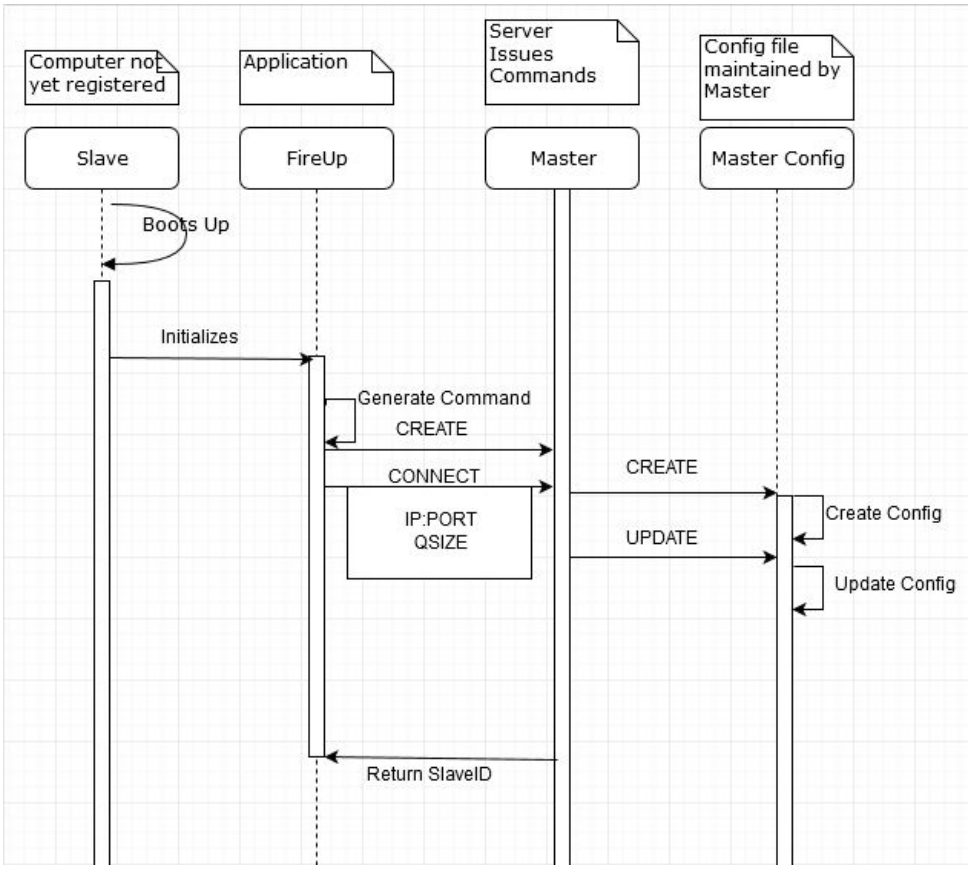

fig.2 StartUp Sequence

REQUIREMENTS TO MEET FOR SCENARIO

R 1.1.1 – Creation of ‘FireUP’ module . Fire Up’s User Manual.
R 1.1.2 - Request and procuring the config file through ‘FireUp’.

ii)BOOTING UP OF A3 BATCH SLAVES

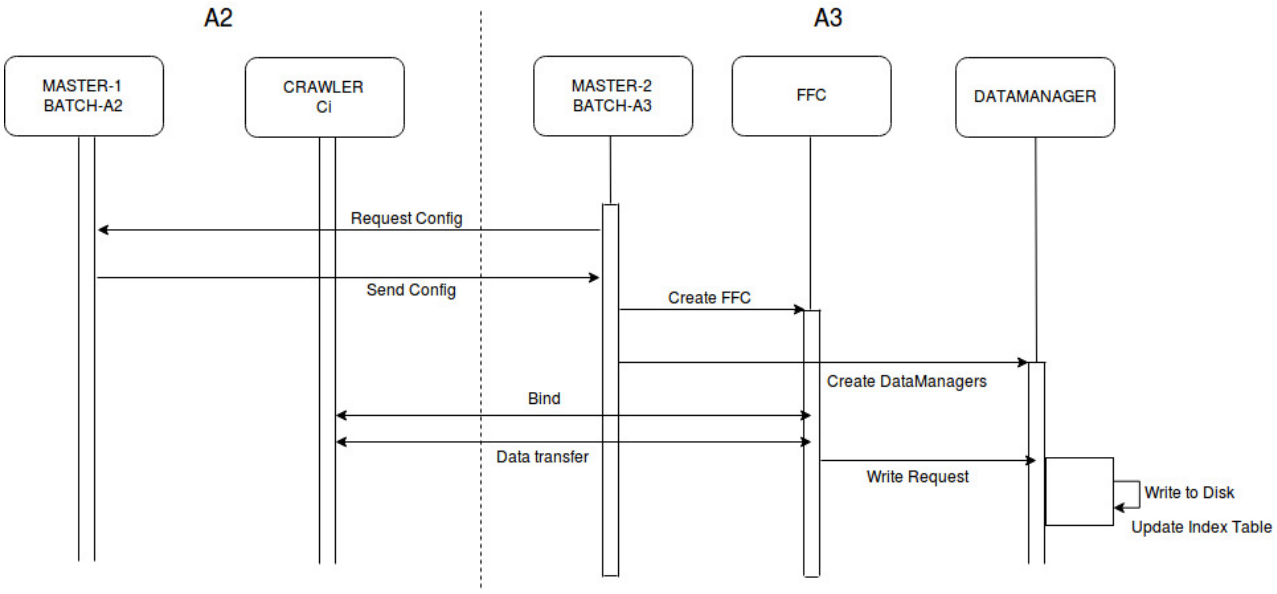

fig.3 Boot-up of slaves (FFCs and DMGRs)

REQUIREMENTS TO MEET FOR THE SCENARIO

R 1.1.3-Booting up of FFC and DataManager modules in accordance with configuration.
R 2.1-Fetching data from the crawler.
R 2.3-Send data to DataMangaer.

iii) Design of Hash Table

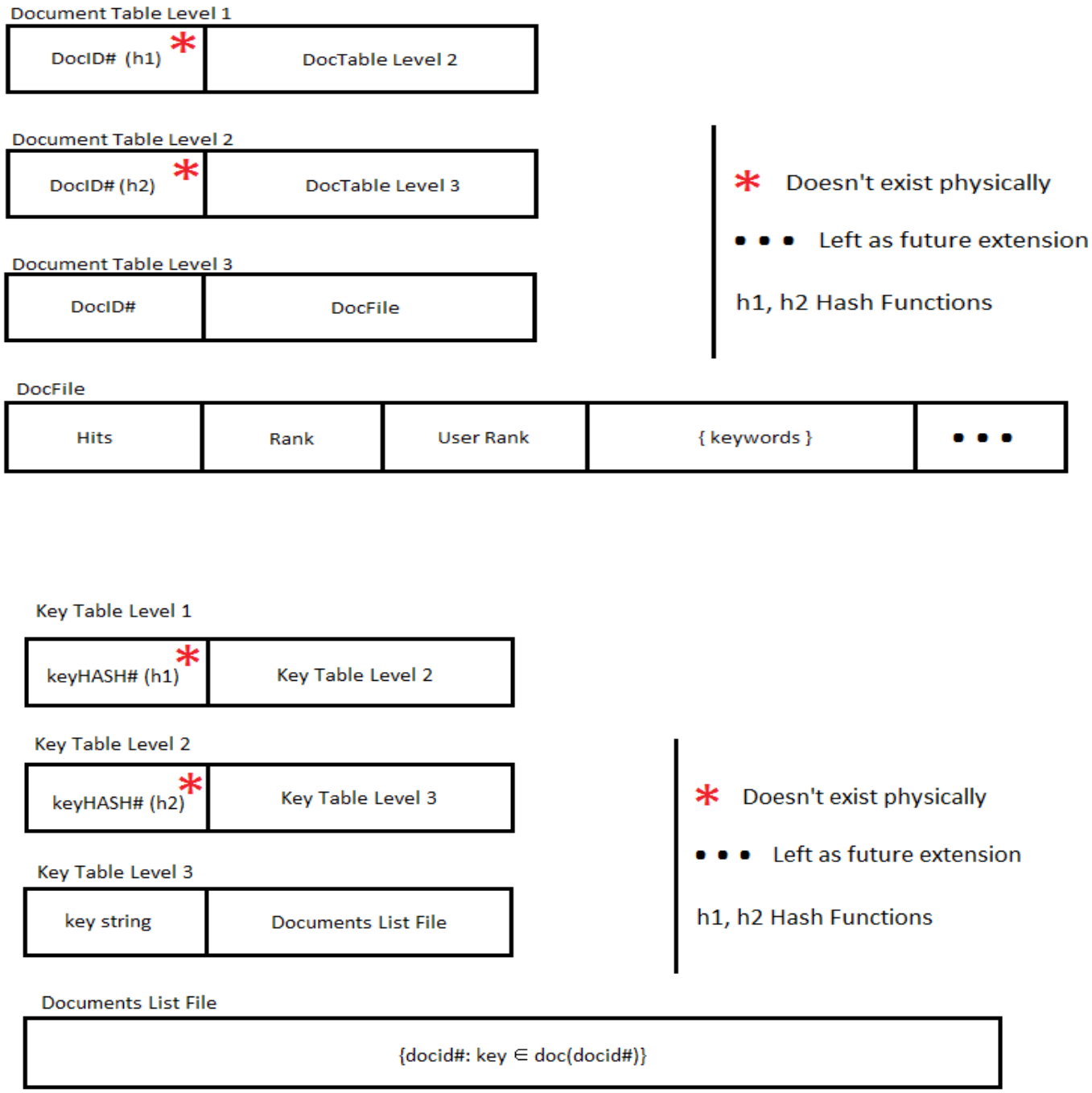

fig.4 Preliminary design of Index table

REQUIREMENTS TO MEET FOR SCENARIO
R3.1- Effecient datastructure to get "***key*-> *links*"**

iv)Query Processing and Result Generation:

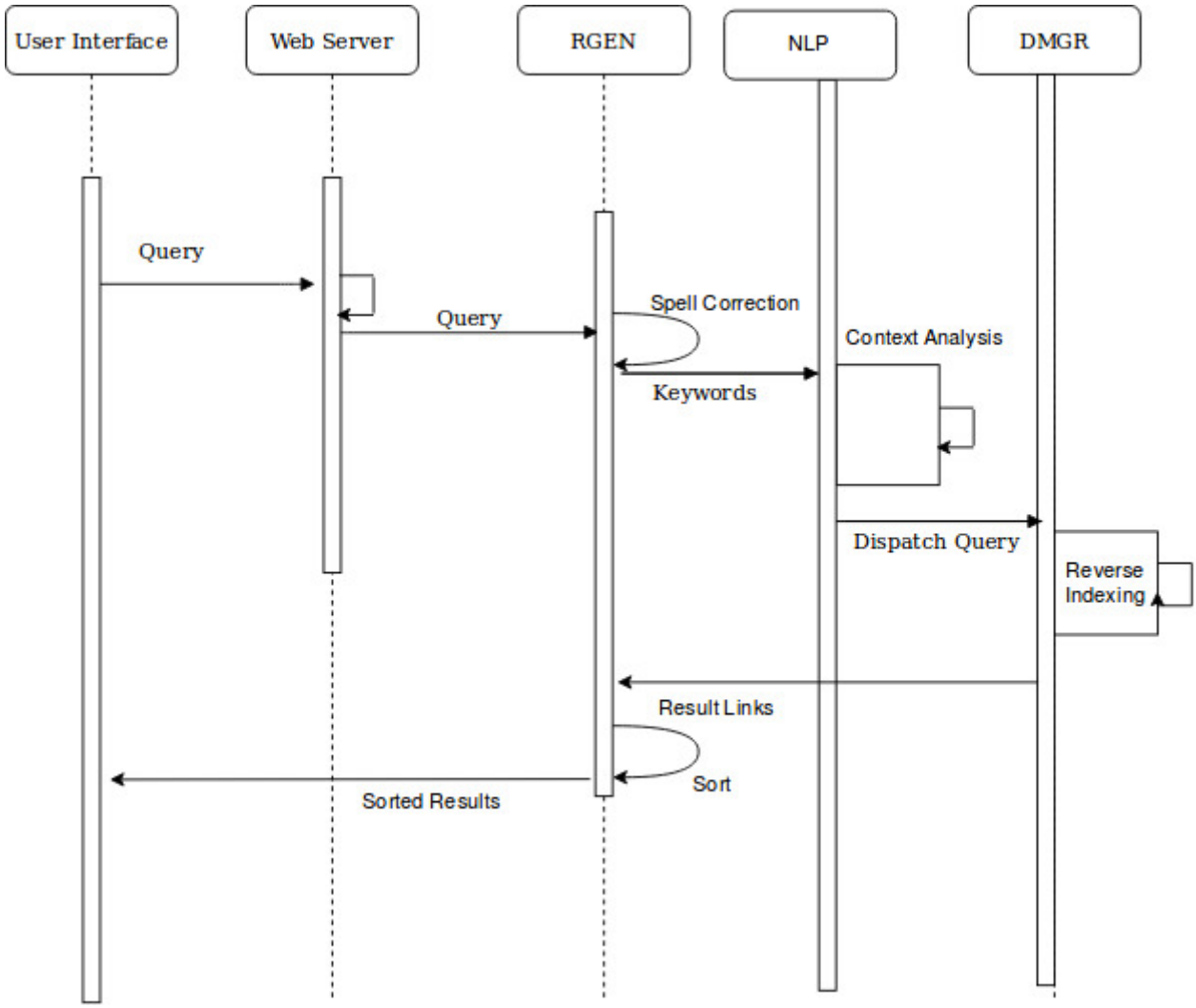

fig.5 Query processing and result generation

REQUIREMENT TO MEET FOR SCENARIO :
R4.1.- Query processing involves spell check,context analysis and splitting to keywords
R5.1-Receving links from datamanagers
R5.2-Sorting links based on their rank.
R6.1-Forwarding results to UI

MODULE DESIGN:

1.Master:

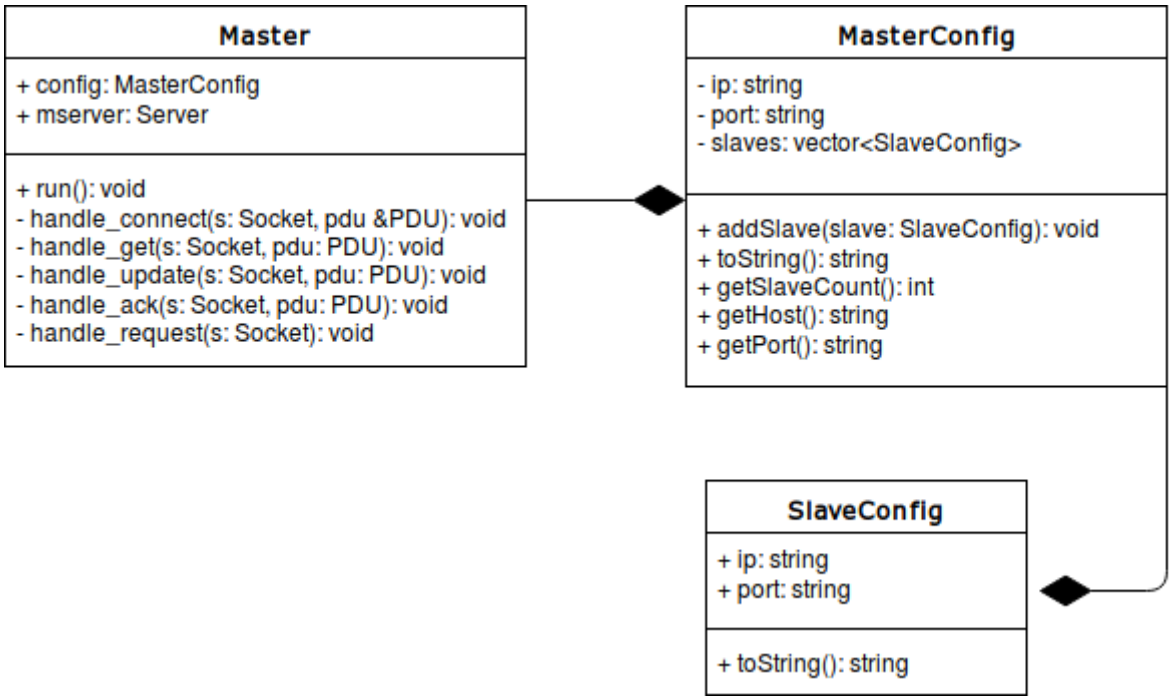

fig.6.1 class diagram Master A3

2.Fireup:

Fireup classes

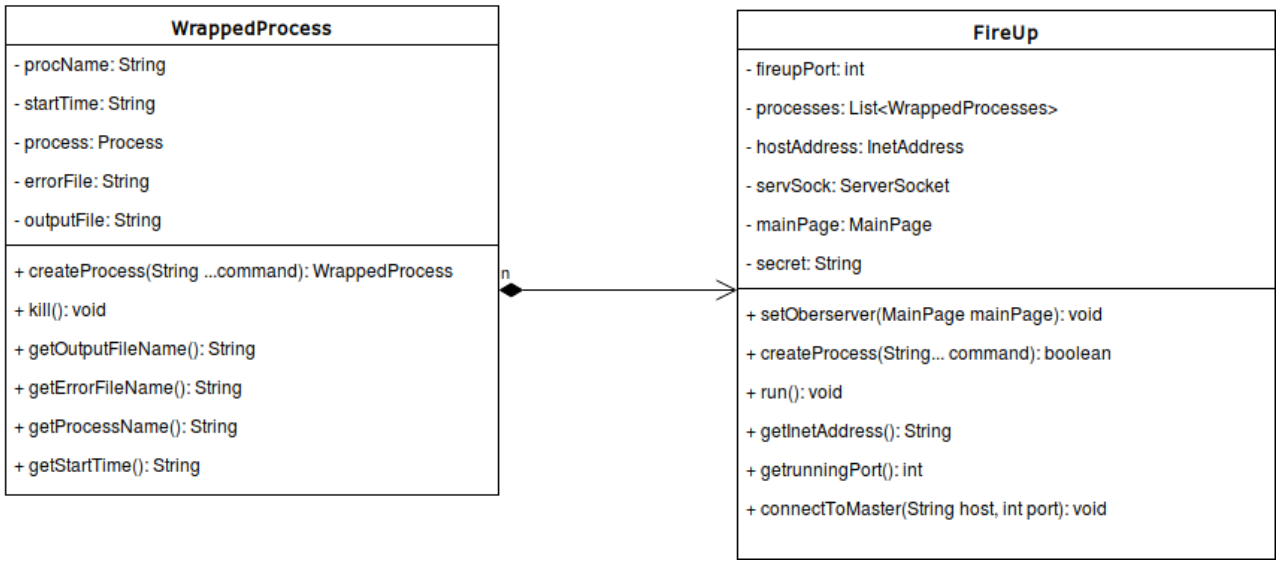

fig.6.2 class diagram FireUP

3.RGEN:

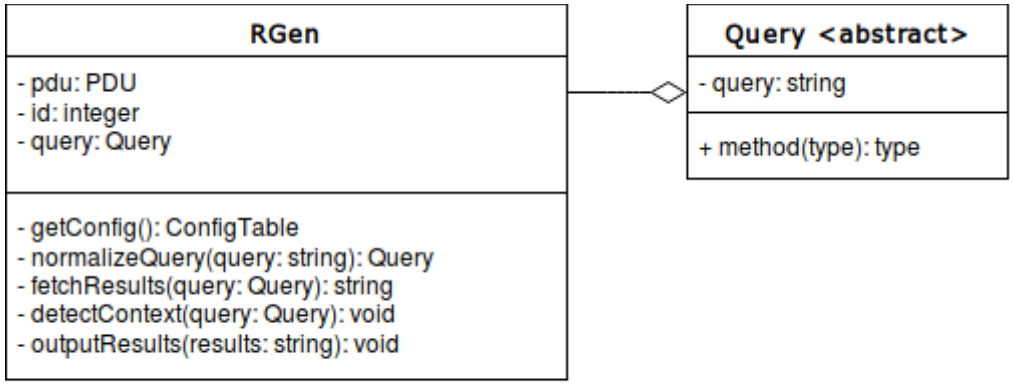

fig.6.3 class diagram RGEN

4.DMGR

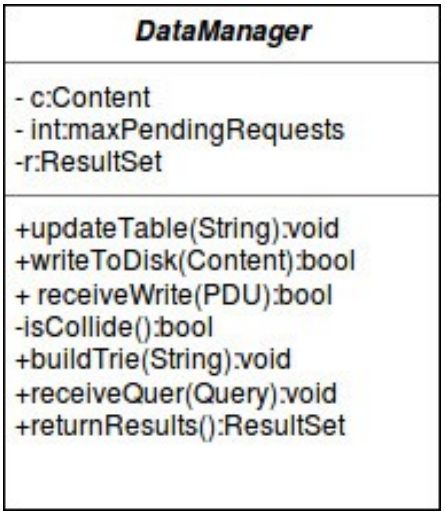

fig.6.4 class diagram DMGR

User manual for `fireup`.

Need for `fireup`.

The SE system we were developing was distributed and someone was needed for managing processes in the machines that were involved. The human operators are one of the inefficient way doing it as it requires to remember a lot of commands and arguments which is quite difficult. In the way to ease this process we developed a standalone application called `Fireup` in Java.

`Fireup` is an application used as a proxy to human operator for our SE system. It provides a programmable interface to manage processes in the machines that it runs (ofcourse with some privilages). `Fireup` also provides the GUI to allow user to manually operate it.

Prerequisites.

1. Java

How to use.

1. Booting up

Just run the command to start `fireup` from your commandline.

```
java –jar fireup.jar
```

2. Running programs:

After start you can run the programs of your choice in your local machine by specifying them in the textbox `executable` with conmmandline arguments in the `cmd args` textbox. [ofcourse hit Run]. First thing in SE you wish to run is your master. So choose the application and run it with the required args.

3. Registering to a master.

`Fireup` support a protocol where you can register your machine as a slave to master machine. By which you give previlages to the master to run commands on that machine. The registration process is manual hence you need to know the hostname/ip and port on which master is running. Enter the fields and hit enter.

4. Whoa! You are ready to acccept commands.

Now your `fireup` is up and running and your master can specify commands to execute.

The programming interface.

The `fireup` provides a simple programming interface where a master can specify commands to execute on the machine. This requires the `fireup` to be registered to the master which is done done in previsous step.

The `fireup` uses ***json*** as a standard to communicate as it's easy to read and understand. The fields of the protocol is given in below table.

1. Connect request.

The connect request is sent by the `fireup` to the master with the the hostname/ip and port on which `fireup` is listening. You need these to communicate with `fireup`, so better save them. This request also sends you a `key` which is used for security purposes. You need to save this as you are supposed to send this key in next communications with `fireup`.

2. Commands

The commands are sent by the master to `fireup` to execute. This includes 4 important fields

a. Key //secret key given to you
b. Command //one of the commands described below.
c. Executable //executable file name/path.
d. Arguments //arguments to the executable.

Fields in the PDU.

Field	Possible Values
Key*	A number 65536
Command	CREATE, KILL,CONNECT
Executable	A valid executable file path
Argument	A string of arguments
Host	Host on which `fireup` is running
Port	Port on which `fireup` is listening.

Table -3. FireUP arguments

NR.1 Protocol used by modules to communicate.

The modules in the system need to communicate with others inorder to resolve their data-dependencies, eg. *FFCs need to communicate with the crawlers to get the web-content.* We introduce a simple protocol to communication between the modules. Which is discussed below. Each PDU is in the JSON format to ease the parsing process. The fields in the PDU are discussed below.

Table-4. PDU fields

Parameter	Description	Possible Values
method	what do you want	GET, CONNECT, CREATE, UPDATE, KILL, ACK
receiver_ip	the ip of host this packet is being sent to	valid ip address
receiver_port	the port on which the peer is listening	valid port number (not a string)
sender_ip	the ip of the sender host	valid ip address
sender_port	the port from where you are sending this PDU	any valid port number
who	the class name of sender	CRAWLER, MASTER, FFC, DMGR, WS
whom	the class name of receiver	CRAWLER, MASTER, FFC, DMGR, WS
data	data needed for processing the request	any valid json string

```
// a sample PDU.
{
    "method": "GET",
    "receiver_ip": "123.231.212.100",
    "receiver_port": 3475,
    "sender_ip": "122.123.124.135",
    "sender_port": 2343,

    "who": "crawler",
    "whom": "master",

    "data": "a_json_string"
}
```

User Interface Design Document.

The user interacts with the system using user interface. This document enumerates the design aspects of the UI. There are two parts of the UI which are needed

1. Search Interface.
2. Monitering Interface.

Search Interface.

This is provided to the user through a web-browser. It allows the user to search documents on the web and view them by giving access to the docs through hyperlinks. Below is the prototype of the UI.

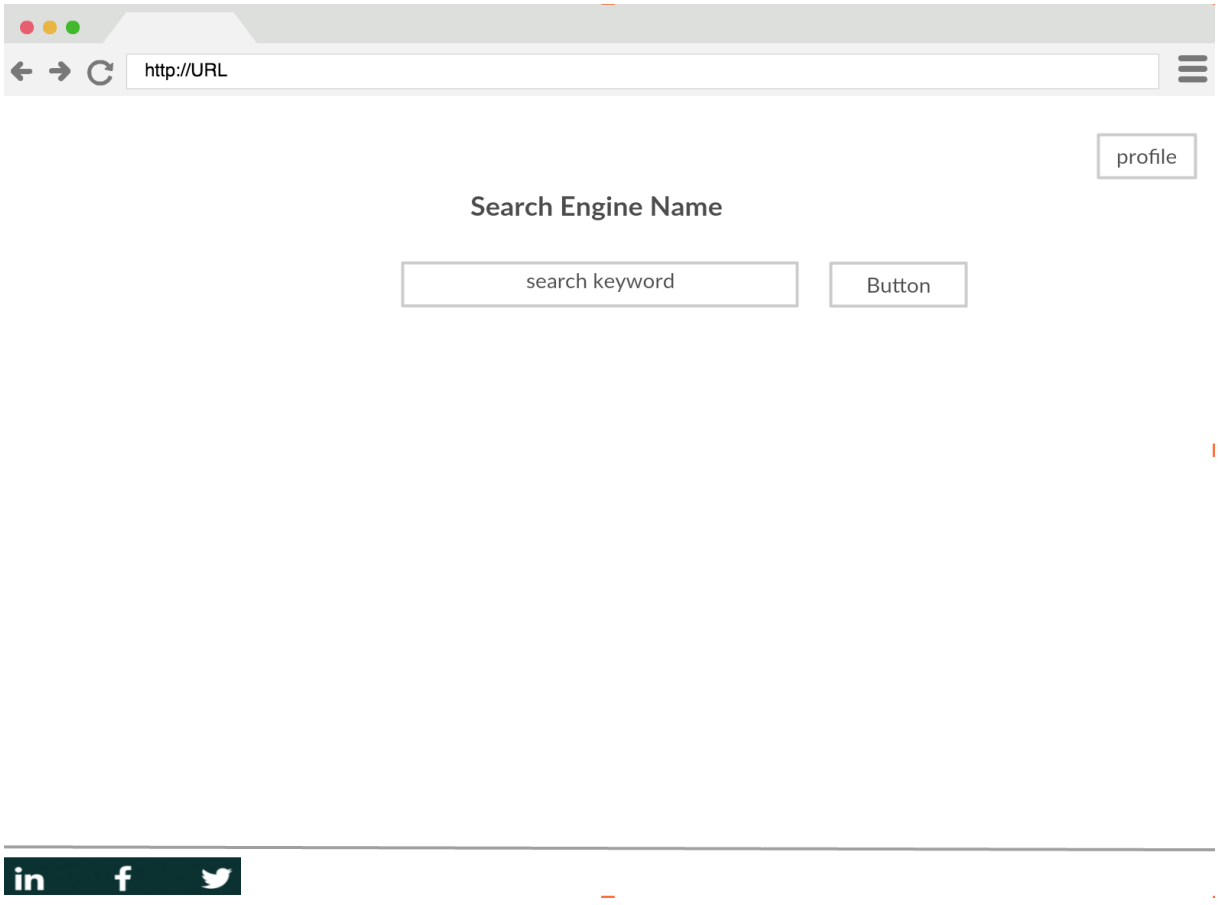

Figure 1. Search Page.

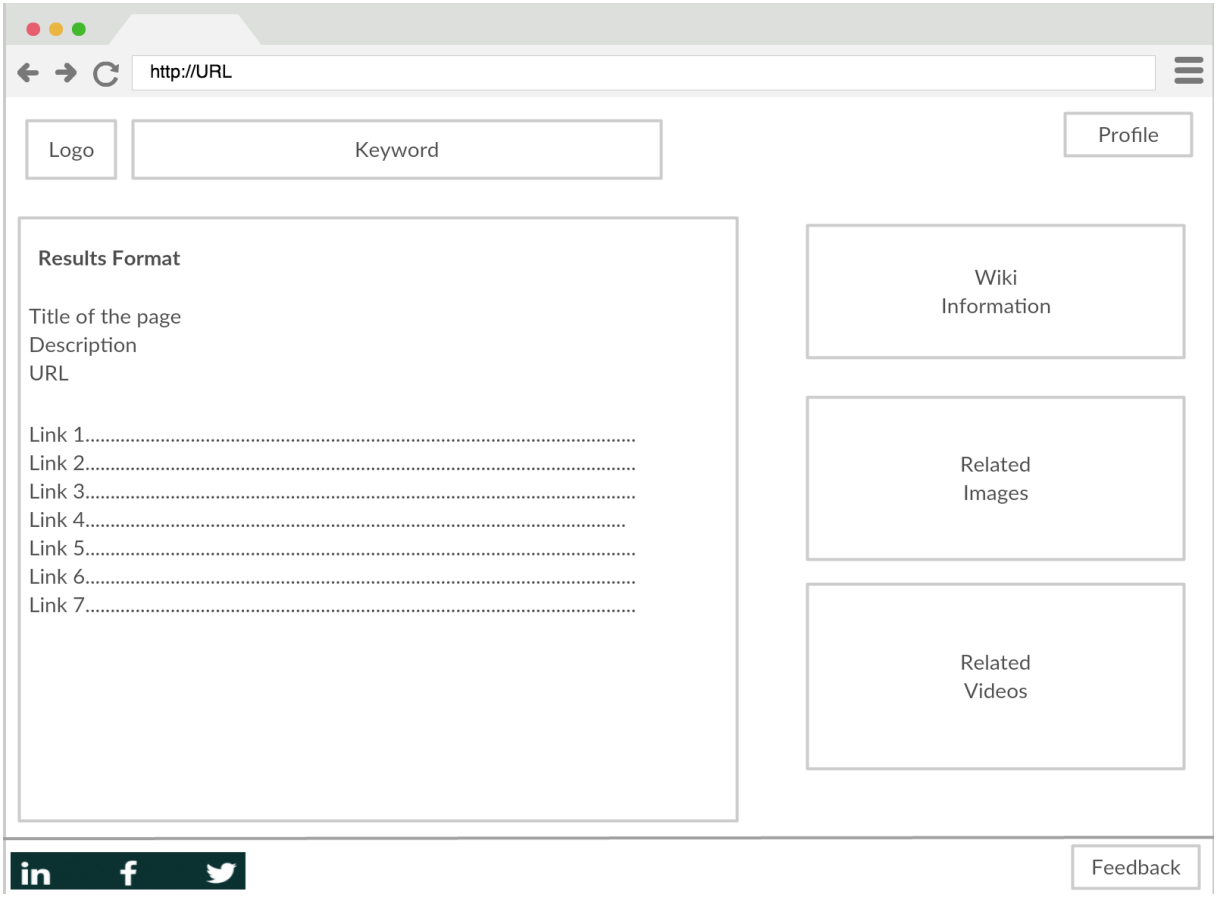

Figure 2. Results Page

How the search works.

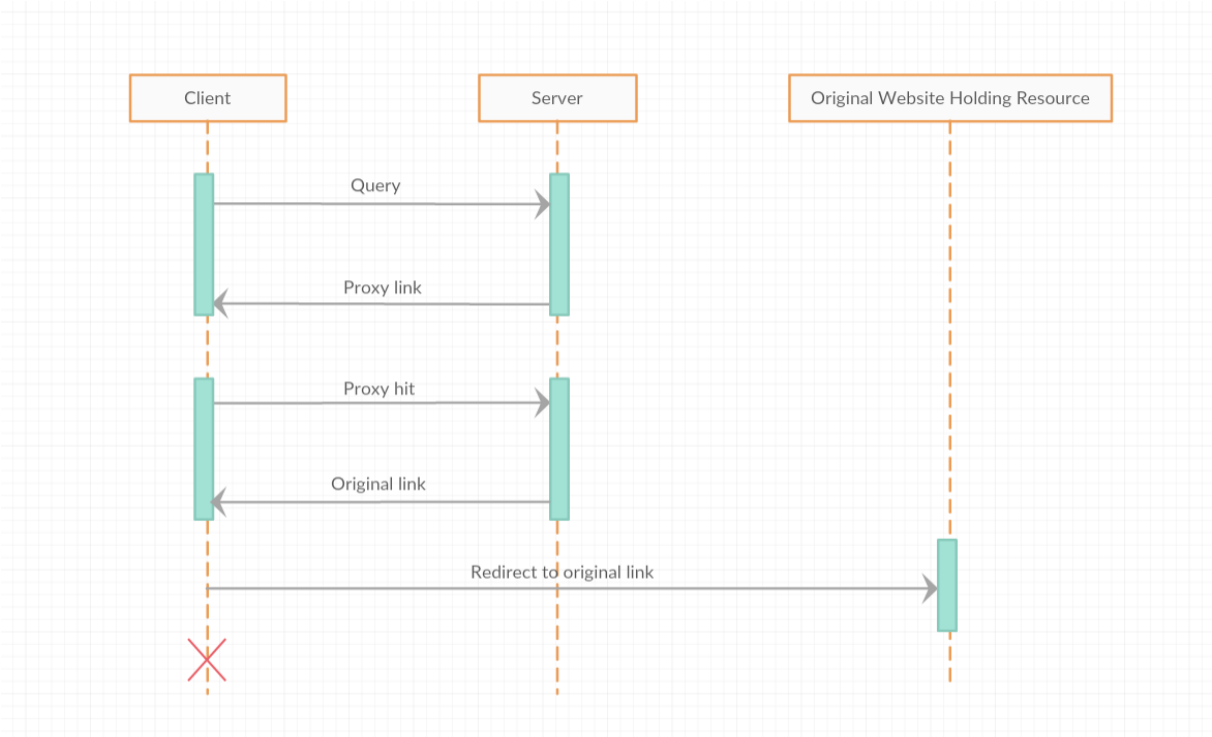

Figure 3. Search process.

UI for Monitering processes.

Requirements.

R1. A software to monitor the processes running on different machines by which we can maintain and debug the system.

R1.1 The system needs to probe the processes periodically and list their status.

R1.2 Allow the user to set the frequency.

Probing PDU.

Request:

```
{
     “method”: “GET”,
     “resource”: “status”,
     ...
}
```

Response:

```
{
     “status”: “UP”,
     “starttime”: “30 Nov 2017 21:34:45”,
     “pid”: 123,
     “data”: “some extra info for further extension”,
     ...
}
```