Introduction to

Sendic Search Engine 0.1

DongChul Park

2009.9

본 문서는 2009년 8월에 개발된 Sendic Search Engine 0.1 에 대한 대략적인 소개 문서입니다.

기존에 SENDIC 1.02 프로그램은 단일 프로그램으로 제작 되었으며 앞으로 업그레이드 될 SENDIC 2.0은 새로 개발된 Sendic Search Engine 0.1 버전이 사용 되었습니다.

좀 더 자세한 기술적인 내용은 향후 작성될 논문 형식의 R&P 기술 문서에서 공개될 예정입니다.

- 박동철 (A Gun)

CONTENTS

1. Overview
2. Index Model / compression
3. Data Model
4. SSE Layer
5. Function
6. Sample Code
7. Conclusion

OVERVIEW

Sendic Search Engine 란?

기존의 검색엔진 시스템이 Server 환경에서 big size data 를 다루는 반면에 SSE 0.1(Sendic Search Engine) 는 Application 환경에 맞는 data를 다룰 수 있는 개방형 검색 엔진이다.

현재 C#으로 구현 되어 있으며 향후에는 다른 language 로 구현될 예정이다. 또한 application 버전 이외에도 server 환경 버전도 개발할 계획이다.

SSE는 비 상업적인 연구 목적이기 때문에 추후에 Open Source Project 로 전향할 계획이다.

OVERVIEW

SSE version 0.1 issue

0.1 버전은 아래와 같은 항목들이 주요 연구 목적이었다.

1. 기존의 Inverted Index 와 다른 모델이 필요.

   - inverted index 모델은 document 와 term (word) 단위로 indexing 한다. Application 환경에서는 inverted index model 보다 data 에 대한 detail 한 access 가 가능 해야 한다. 또한 원본 data 의 유실 없이 indexing과 동시에 compressing 기능이 되어야 한다.

2. basic data model 과 basic search function 기능을 구현

   - 0.1 은 많은 기능을 구현 하는 것 보다 1번 항목을 바탕으로 basic data model의 구현과 기존 기능의 구현을 목표로 한다.

INDEX MODEL

Constituent Unit

아래와 같이 3가지로 구분된다.

1. Term (Word)

   -> index 을 구성하는 기본 단위

2. Sentence

   -> 한 라인 단위의 Term 들의 집합

3. Semantic Data
   -> Sentence 단위로 부여 할 수 있으며 tagging, metadata, Translated Data 등으로 사용할 수 있다.

INDEX MODEL

File Format

Index 생성할 경우 3가지 파일이 생성되며 각각의 format 은 아래와 같다.

1. filename.index (index of word)
   -> [length of word][length of number in Word][number of word][word]...
   -> size : 1byte + 1byte + below 4byte + length of word * 1byte

2. filename.sen (index of sentence)

   -> [number of word in sentence][length of word index][group of word index][word index]...
   -> size: 1byte + 1byte + 1byte + below 4byte

3. filename.sem (Semantic Data)

   data ASCII(10)..

INDEX MODEL

Indexer

SSE ver 0.1 에서는 SIndexer 가 구현 되어 있으며 SIndexer 는 RIndexer와 WIndexer 을 포함 하고 있다.

1. SIndexer (Standard Indexer)
   - 기본적인 indexing 기능을 구현하였다. RIndexer와 WIndexer를 이용하여 기능을 구현한다.

2. RIndexer (Read Indexer)
   - Index data를 읽는다. (memory or file)

3. WIndexer (Write Indexer)
   - Index data 를 생성하여 저장 한다 (memory or file)

DATA MODEL

SSE 0.1 의 data model 은 Sentence Unit 과 Term Unit 으로 구분된다.

Term Unit

1. Word
   - Index data를 읽어 오는 기본 단위 이다. word byte 값과 index , group 를 저장한다

2. DTerm
   - index 생성시 word data를 저장할 때 사용되는 기본 단위, word byte 값과 group 값을 저장

3. CTerm
   - DTerm 에 대해서 counting 연산에 대한 추가 정보를 저장한다.

4. GTerm
   - Cterm을 group 단위로 관리 한다.

COMPRESSION

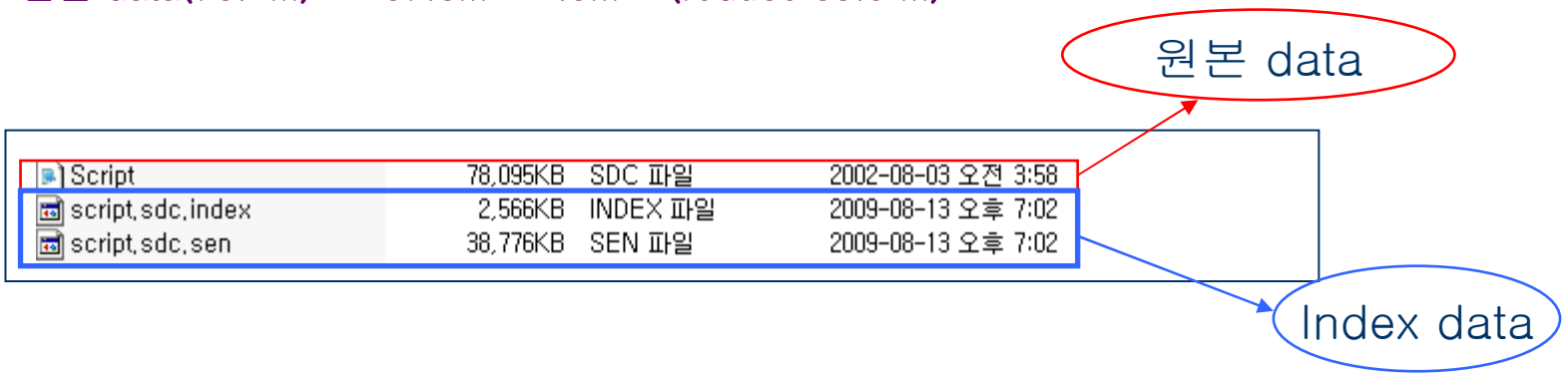
Gigabytes 를 다루는 server 환경과는 달리 일반적인 application program 환경에서는 megabyte 단위의 용량도 민감하게 반응한다.

SSE Engine 을 적용해야 할 SENDIC 2.0 PROGRAM 에서도 script data 가 76.2 M 이며 해당 data의 용량을 필요한 data 의 유실 없이 compressing 하는 것이 이번 version에 중요한 issue 였다.
구현 당시 compress 관련 여러 algorithm을 고려 했으며 대표적인 algorithm으로는 zip, variable byte codes, Huffman code (with Huffman Tree) 등 이었다.

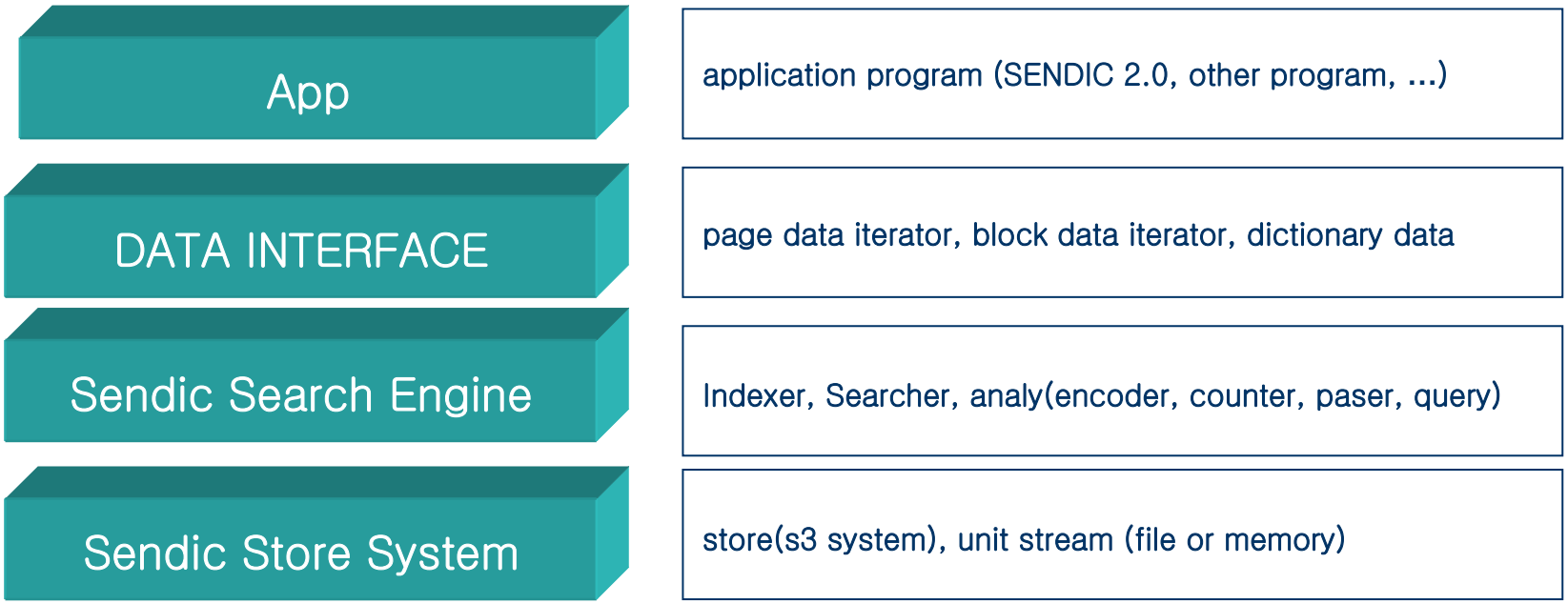
여러 가지 테스트 결과 zip 계열 algorithm 은 decode 관련 시간이 search 실행 시간에 많은 부담을 주어 seach 기능에 적용하기에는 맞지 않는 algorithm 이었고 huffman code algorithm 은 테스트 결과 huffman tree의 사용 bit 가 7bit 로 계산 되어 구현 하는 비용에 비해 비효율적으로 판단 되었다.

실질적으로 index format 에서 word index 값과 number of word 값을 제외한 항목은 byte 값을 사용 하기 때문에 variable byte codes 방식을 구현할 경우 구현한 복잡도에 비해서 비효율적인 결과로 나타났다. 하지만 data size 가 커질수록 variable byte codes 방식이 효율적일 것으로 기대 된다.
이번 version 에서는 variable byte codes 방식을 적용하지 않고 간단하게 응용한 방식을 적용하였다. word index 와 number of word 항목의 사용하지 않는 byte를 체크하여 사용하지 않는 integer byte 의 size 를 줄이는 방식이다. 또한 word 들을 group 으로 나누어 관리 하여 word index 의 size 도 감소 시켰다. 이런 방식을 적용시킴으로써 indexing 과 동시에 compression 을 적용 하였다.

원본 data(76.2M) => 37.8M + 2.5M (reduce 35.9 M)



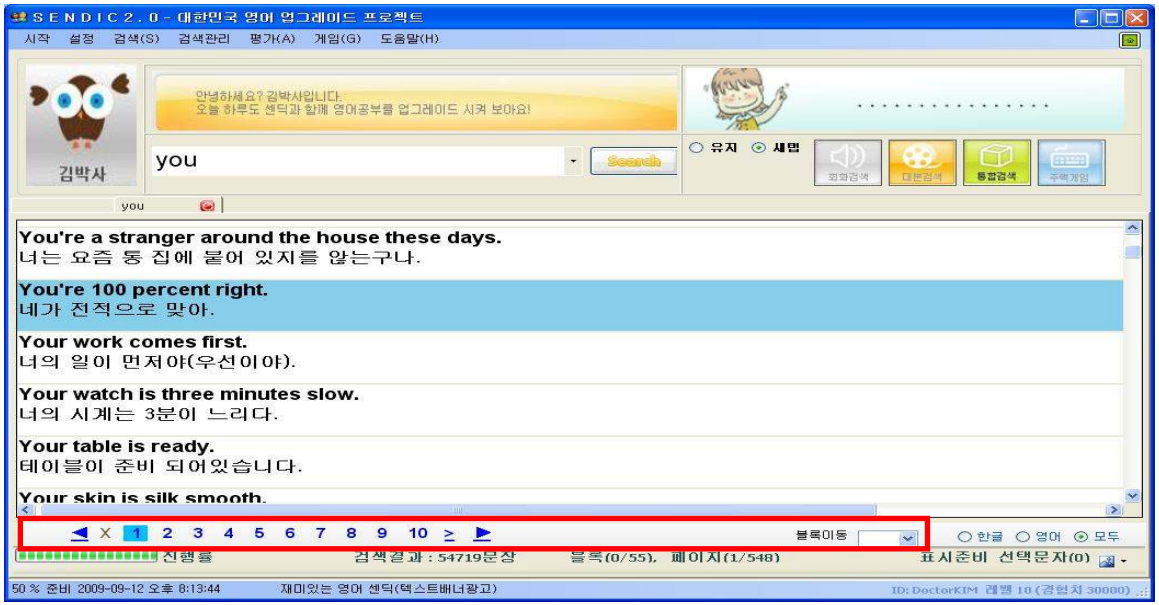
SSE LAYER


App
application program (SENDIC 2.0, other program, ...)
DATA INTERFACE
page data iterator, block data iterator, dictionary data
Sendic Search Engine
Indexer, Searcher, analy(encoder, counter, paser, query)
Sendic Store System
store(s3 system), unit stream (file or memory)

FUNCTION

Page Access Iterator

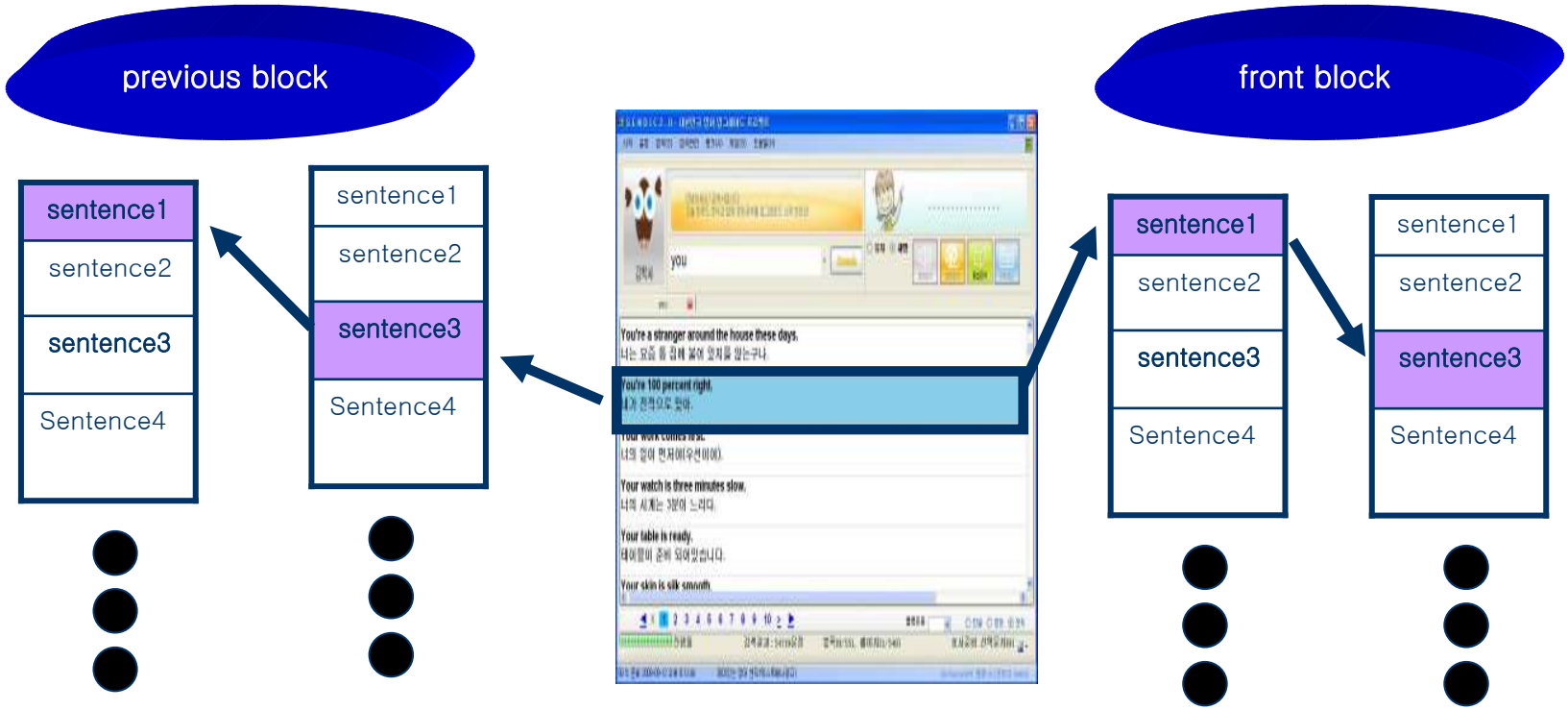
검색된 결과를 page Access Iterator 을 이용하여 Page 단위로 접근 할 수 있다.



FUNCTION

Block Access Iterator

검색된 결과를 Block Access Iterator 을 이용하여 Block 단위로 접근 할 수 있다.
검색된 결과에서 선택된 sentence를 기준으로 원본 data의 previous block 이나 front block로 jump 해서 data를 선택할 수 있다. jump Block 는 순차적이고 연속적이다.

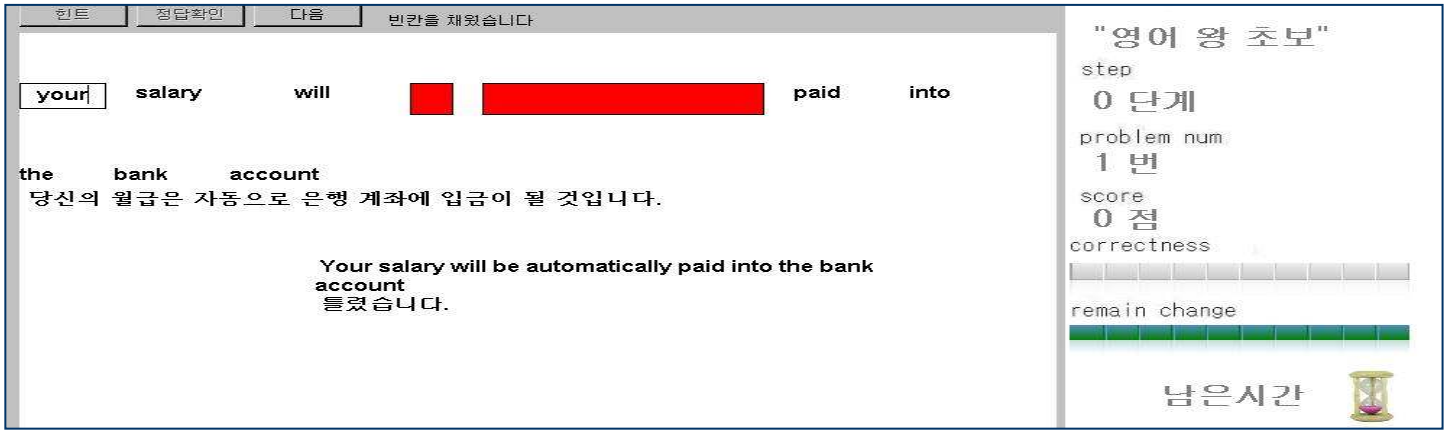


FUNCTION

Select Sentence

지정된 number 의 word 로 구성된 sentence 를 검색 할 수 있다.

아래 그림과 같이 level 별로 sentence 를 구성하는 word 의 number를 변화를 시켜서 검색한 sentence 을 가지고 game 에 적용 할 수 있다.



SAMPLE CODE

SSE 0.1 C# 버전은 dll 형태나 Class Library 형태로 제공 된다.

자신이 개발하고 있는 프로젝트에 dll 이나 library 를 추가 한 후 자신에게 맞는 application 기능을 구현 하면 된다.

SAMPLE CODE

Indexing

Sindex 를 이용하여 자신이 원하는 text file 을 index file 로 변환하여 구성 할 수 있다.

[Make Index Code]

```
Sindex sindex = new Sindexer(); // create sindexer instance.
sindex.Make("test.txt"); // make index, using test.txt.
Sindex.Close();
```

[Make Index Code with FSUnit]

```
FSUnit funit = new FSUnit(); // create FSUnit instance.
funit.open("test.txt"); // open test.txt file.
SIndex sindex = new Sindexer(funit);   // create sindex with FSUnit.
sindex.Make(); // make index using FSUnit.
Funit.Close() // Or sindex.Close().
```

[Make Index Code with Semantic Data]

```
Sindex sindex = new Sindexer(); // create sindexer instance.
sindex.Make(true, "test.txt"); // make index using test.txt And make semantic data.
Sindex.Close();
```

SAMPLE CODE

SSeacher

SIndexer 로 구성한 index data 를 SSearcher 를 이용하여 검색할 수 있다.

[Search “apple”]

```
FSUnit fu = new FSUnit();  // create FSUnit instance, using file system.
SIndex sindex = new SIndexer(fu); // Create Sindexer.
SSearcher search = new SSeracher(“test.txt”); // create SSearcher instance and
                                              //      setting search file name.
Query query = new Query(“apple”);           // create Query instance and setting query string.
search.Search(sindex, query); // search “apple”, using standard indexer and standard query.
```

SAMPLE CODE

Page Iterator

SSearch로 검색한 data 를 Page Iterator 을 이용하여 page 단위로 sentence 를 access를 할 수 있다 .

[Print string of sentence in page]

```
...    search “apple code”   ...
search(sindex, query);

Page rpage; // define Page Iterator.
WSentence wsen; // define sentence buf.
byte[] data; // define byte data buf.
string str;
rpage = search.GetPage(1); // get data in first page.

While((wsen = rpage.GetCurData()) !=null)
{
    data = wsen.GetSenByte();  // get bytes of sentece.
    str = System.Text.ASCIIEncoding.ASCII.GetString(data); // convert bytes into string.
    Console.WriteLine(str);
}
```

SAMPLE CODE

Block Iterator

SSearch로 검색한 data 를 Block Iterator 을 이용하여 Block 단위로 sentence 를 access를 할 수 있다 .

[Print string of sentence in pre block]

```
...    search “apple code”   ...
search(sindex, query);
Page rpage; // define Page Iterator.
Wsentence[] wsens; // define sentence buf.
byte[] data; // define byte data buf.
string str;
SBlock block; // define Block Iterator.
SPart part;  // define data Iterator.
rpage = search.GetPage(1); // get data in first page.
block = search.GetBlock();  // get Block Iterator.
part = block.GetBlock(3);   // get second data iterator.
wsents = part.GetPreBlock();     // get previous block of second data.
for(int i=0; i<part._PreLen; i++)
{
    data = wsens[i].GetSenByte();  // get bytes of sentece.
    str = System.Text.ASCIIEncoding.ASCII.GetString(data); // convert bytes into string.
    Console.WriteLine(str);
}
```

Conclusion

SSE 0.1 에서는 앞부분의 issue에서 작성한 것처럼 기본적인 data model 과 index model 을 구현하는 것을 목적으로 하였다.

계속 개발되고 있는 SSE 0.2 에서는 similar search, multi field search 등과 같은 고급 검색 기능이 구현되며 0.1버전에서 일부 구현 된 Virtual Sendic Store System 을 더욱 강화 시켜 향후 계획하고 있는 Network 기능과 유연하게 연동 할 수 있도록 설계할 계획이다.

SSE 0.2 issue

- Advanced Search Function.
- Virtual Sendic Store System. (for distributed process)
- Design Open Architecture.

Reference

N ISLAND : finecode.net

EMAIL : pdc222@naver.com