Implementasi Algoritma *Ternary Search Tree* dan Teknologi Grafis Berbasis Vektor untuk Interpretasi Alfabet *Pitman Shorthand*

[1] Irwan Sembiring, [2] Theophilus Wellem, [3] Gloria Saripah Patara

Fakultas Teknologi Informasi
Universitas Kristen Satya Wacana
Jl. Diponegoro 52 – 60, Salatiga 50711, Indonesia
Email : [1] irwan@uksw.edu, [2] erman_wellem@yahoo.com,
[3] glo_saripah@yahoo.com

Abstract

Pitman shorthand system is one of the most popular of nowadays shorthand systems. It has been taught and widely used throughout the world to help the stenographers in improving the writing speed. In this research, a Pitman shorthand's alphabets interpretation mobile system is established purposely to ease the stenographers in interpreting the shorthand script into English text. The outlines are interpreted using *dictionary-based approach*. This system can display alphabet list, so that it also can be used by the students who wish to learn this shorthand system. All symbols are rendered with *JSR-226* (*Scalable Vector Graphics API*). The system implements *Ternary Search Tree* (*TST*) algorithm to improve data searching speed. According to the experiments, the system can successfully interpret inputted alphabets up to 71.95%.

Key Words: Pitman Shorthand, Stenographer, Dictionary-based , Approach, JSR-226 (Scalable Vector Graphics API), Ternary Search Tree.

1. Pendahuluan

Pitman Shorthand merupakan sistem penulisan cepat yang dikembangkan oleh Sir Isaac Pitman (1813-1897). Sistem penulisan cepat ini dipublikasikan pada tahun 1837. *Pitman Shorthand* adalah sistem fonetis yang menggunakan simbol-simbol untuk berbagai bunyi dalam bahasa [1]. Pemakaian simbol untuk merepresentasikan bunyi tertentu dimaksudkan untuk mempercepat penulisan suatu kata. Hal ini disebabkan suatu simbol dapat menggantikan pemakaian beberapa alfabet Latin yang sering digunakan dalam suatu kata. Penghematan

inilah yang dimanfaatkan oleh sistem penulisan cepat pada umumnya untuk meningkatkan kecepatan penulisan.

Proses penerjemahan ulang suatu naskah yang ditulis menggunakan simbol-simbol *Pitman Shorthand* dapat dilakukan secara manual oleh stenografer yang menguasai aturan penulisan jenis *shorthand* ini. Namun, proses penerjemahan ulang dapat dilakukan dengan memanfaatkan perkembangan di bidang teknologi komunikasi dan informasi, khususnya teknologi pada devais berukuran kecil seperti *handphone* dan *personal data assistant* (PDA). Selain manfaatnya yang semakin beragam, devais ini juga sangat familiar dengan masyarakat.

Pada tulisan ini akan dibahas sistem penampil dan penginterpretasi alfabet *Pitman Shorthand* menggunakan metode *template-based approach* dengan memanfaatkan *Scalable Vector Graphics (SVG)* untuk menampilkan konsonan, vokal, *diphthong*, *grammalogue* (*short-form*) maupun *intersection*. Sedangkan, untuk proses interpretasi menggunakan metode *dictionary-based approach*. Proses pencarian data pada database *template* dan kamus menggunakan algoritma *Ternary Search Tree (TST)* untuk menekan besarnya waktu pencarian. Sistem ini dapat menjadi solusi untuk proses penerjemahan ulang naskah yang ditulis dengan *Pitman Shorthand* ke teks bahasa Inggris.

2. Kajian Pustaka

Pitman Shorthand merupakan metode penulisan yang dirancang untuk memaksimalkan kecepatan penulisan dan pembacaan kembali [2]. Sistem yang dikembangkan oleh Sir Isaac Pitman (1813-1897) ini menggunakan simbol-simbol untuk berbagai bunyi pengucapan kata dalam bahasa Inggris [1]. Simbol-simbol yang digunakan berbentuk geometris seperti garis lurus, kurva, dan titik.

Terdapat beberapa dialek dari *Pitman Shorthand*, antara lain o*riginal Pitman's*, *Pitman's New Era*, dan *Pitman's 2000*. Versi yang terakhir menghilangkan simbol-simbol tertentu dan memperkenalkan penyederhanaan lain pada versi sebelumnya. Misal, *stroke* "rer" dan "kway" ada pada *Pitman's New Era,* tetapi tidak ada pada *Pitman's 2000* [1]. Sistem yang dibangun menggunakan versi *Pitman's New Era,* karena versi ini masih banyak dipakai walaupun *Pitman's 2000* sudah diperkenalkan.

Alfabet *Pitman Shorthand* yang tidak baku dapat dibentuk menggunakan *Scalable Vector Graphics* (SVG). SVG adalah bahasa yang digunakan untuk mendeskripsikan grafis 2D berbasis *eXtensible Markup Language* (XML). SVG dapat menangani tiga obyek grafis yaitu: bentuk vektor grafis (misal, *path* yang terdiri atas garis lurus, kurva, dan segi empat), citra, dan teks. Obyek-obyek grafis dapat dikelompokkan, ditransformasi dan dibentuk berdasarkan obyek yang sudah didefinisikan sebelumnya. Grafis yang dihasilkan lewat SVG bersifat interaktif dan dinamis [3].

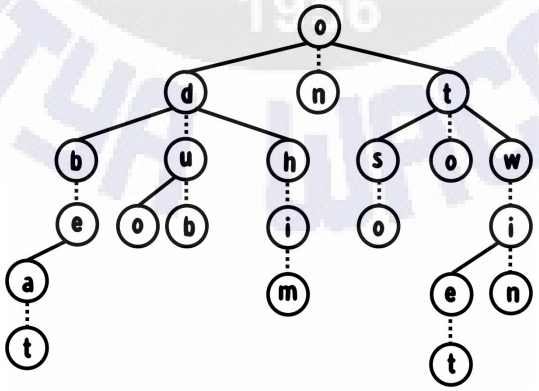
SVG 1.1 mempunyai dua *profile* yaitu: SVG *Tiny* (SVGT*)* dan SVG *Basic (*SVGB*)*. *Profile* pertama difokuskan pada *handphone,* sedangkan *profile*

kedua untuk devais yang memiliki spesifikasi lebih tinggi seperti *Personal Data Assistant* (PDA). Beberapa faktor pada *mobile device* seperti keterbatasan *memory*, daya CPU, dan tampilan mengakibatkan kedua *profile* ini mendukung fitur-fitur yang terbatas juga.

Pencarian data *template* yang berhubungan dengan proses penciptaan alfabet menggunakan algoritma *Ternary Search Tree*. *Ternary Search Tree* merupakan suatu *ternary tree* yang menggabungkan efisiensi waktu (*time efficiency*) dari *digital trie* dengan efisiensi tempat (*space efficiency*) dari *Binary Search Tree* (BST). Struktur yang dihasilkan lebih cepat jika dibandingkan dengan *hashing* untuk kasus-kasus tertentu [1]. Salah satu kasus tersebut adalah pencarian data yang tidak ada di dalam *tree* (*unsuccessful searches*). Berdasarkan artikel yang ditulis oleh *Jon Bentley dan Bob Sedgewick*, algoritma ini hanya memerlukan seperlima dari total waktu yangdiperlukan oleh *hashing* untuk kasus *unsuccessful searches* [4].

Digital trie menyimpan data berupa koleksi *string*. Masing-masing *node* pada *trie* merepresentasikan karakter awal dari suatu *string* dan bercabang sebanyak *N* buah, dimana *N* merupakan jumlah karakter *string* dikurangi satu. Jika nilai dari *N* terlalu besar, maka jumlah *node* juga akan menjadi besar. TST lebih mudah dipahami jika direfleksikan sebagai suatu *digital trie* yang *N* buah *node*-nya digantikan dengan *binary tree*. Selain *left node* dan *right node*, terdapat *middle node* yang berhubungan dengan karakter yang dirujuk oleh *parent node*.

Algoritma ini sering digunakan untuk aplikasi kamus, dimana terdapat himpunan *string* yang cukup banyak sebagai kata kunci dan masing-masing kata kunci tersebut mempunyai pasangan nilai. Himpunan kata kunci akan digunakan untuk membangun *tree*, sedangkan pasangan nilai masing-masing kata kunci akan diasosiasikan pada *node-node* yang bersesuaian. Pada proses pencarian, kata kunci akan digunakan untuk memperoleh pasangan nilainya.



Gambar 1 *Ternary Search Tree*

Metode untuk menyusun *left node* dan *right node* mengikuti aturan *Binary Search Tree*, yaitu nilai kata kunci *left node* lebih kecil dari *parent node* dan nilai kata kunci *right node* lebih besar dari *parent node*. Perbedaannya,

setiap *node* pada TST berisi sebuah karakter sebagai ganti *string*.

Middle node akan diisi dengan sebuah karakter yang diambil dari karakter kedua *string* pada koleksi. *String* tersebut diawali oleh karakter pada *parent node*.Jika terdapat lebih dari satu *string* yang memenuhi syarat dan mempunyai karakter kedua yang berbeda-beda, maka elemen tengah dari daftar karakter kedua diisi pada *middle node*. Elemen sisa ditempatkan pada sisi kiri dan kanan.

Pada Gambar 1 ditunjukkan sebuah TST yang merepresentasikan 11 kata yaitu: bat, be, do, dub, hi, him, on, so, to, wet dan win.

Algoritma untuk membangun *tree* dapat disusun dalam langkah-langkah berikut:

- **Langkah 1**, ambil karakter pertama pada setiap kata kunci. Karakter-karakter ini bersifat unik (tidak boleh sama) dan diurutkan. Dari Gambar 1 diperoleh karakter-karakter b, d, h, o, s, t dan w.
- **Langkah 2**, ambil karakter ke-[*N*/2+1] dari daftar karakter (karakter 'o') dan isi pada *root node*. *N* adalah jumlah karakter pada daftar.
- **Langkah 3**, ambil karakter ke-[*N*/2+1] pada sisa karakter di bagian kiri (karakter ke-1 sampai ke-*N*/2) dan bagian kanan (karakter ke-[*N*/2+2] sampai ke-[*N*-1]). Simbol *N* adalah jumlah karakter pada masing-masing bagian. Karakter ke-[*N*/2+1] pada bagian kiri (karakter 'd') diisi pada *left node* dan karakter ke-[*N*/2+1] pada bagian kanan (karakter 't') diisi pada *right node*.
- **Langkah 4**, jika karakter pada daftar karakter sudah semuanya dimasukkan pada *tree*, dilanjutkan dengan memasukkan karakter berikutnya dari kata kunci. Karakter ke-*i* diisi pada *middle node* dari *node* yang berisi karakter ke-[*i*-1] dari kata kunci, dimana *i* = 2, 3, 4, ..., *N*. Pada Gambar 1, karakter 'n' diisi pada *middle node* dari *parent node*-nya untuk kata kunci "on".

Data yang telah diasosiasikan pada suatu *node* dapat dicari di dalam *tree* dengan bantuan kata kunci tertentu. Misal, kata kunci disebut *key*, karakter pada *node* disebut *chr*, data yang dicari disebut *stuff*, *left node* disebut *lokid*, *middle node* disebut *eqkid*, dan *right node* disebut *hikid*. Maka, algoritma pencarian data dapat disusun dalam langkah-langkah berikut:

- **Langkah 1**, inisialisasi : *currNode* = *root*, *N* = 0
- **Langkah 2**, periksa apakah karakter ke-*N* dari *key* memenuhi kondisi lebih kecil, lebih besar, atau sama dengan *chr*. Jika memenuhi kondisi pertama, cari karakter pertama pada *lokid*. Jika memenuhi kondisi kedua, cari karakter pertama pada *hikid*. Jika memenuhi kondisi ketiga, cari karakter ke-*N*+1 pada *eqkid*.
- **Langkah 3**, perulangan dilakukan sampai memenuhi dua kondisi. Pertama, ditemukannya *leaf node* (*currNode* = *NULL*) yang berarti *stuff* tidak ditemukan. Kedua, semua karakter pada *key* telah ditemukan (*N* = *length of key*) yang berarti *stuff* juga ditemukan.
- **Langkah 4**, isi *stuff* dengan data yang diasosiasikan pada *currNode*.

3. Metode Penelitian

Dalam penelitian ini digunakan *Unified Modelling Language* (UML)

sebagai bahasa pemodelan untuk merancang sistem yang akan dibangun. UML adalah bahasa grafis untuk mendokumentasikan, menspesifikasikan, dan membangun sistem perangkat lunak [5].

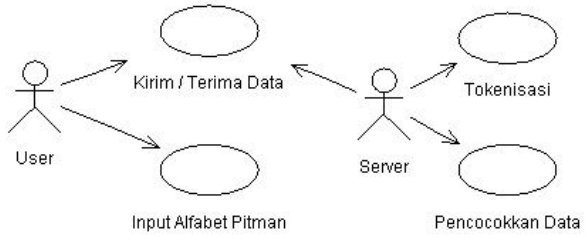
UML terdiri atas sembilan jenis diagram yang dapat digunakan untuk menggambarkan suatu sistem, yaitu: diagram *use-case*, *sequence*, kolaborasi, *statechart*, aktivitas, kelas, obyek, komponen, dan *deployment* [5]. Lima diagram pertama merupakan diagram-diagram yang menggambarkan aspek dinamis dari sistem dan empat diagram yang lain merupakan diagram-diagram yang menggambarkan aspek statis dari sistem. sembilan jenis diagram tersebut seperti ditunjukkan pada Tabel 1.

Pada penelitian ini, sistem dimodelkan menggunakan diagram *use-case*, aktivitas dan kelas. Ketiga jenis diagram ini dipilih karena sudah memenuhi kebutuhan dalam pemodelan sistem yang dibangun.

Tabel 1 Kegunaan Diagram-diagram UML

Diagram	Kegunaan
Use-case	Menunjukkan sekumpulan kasus fungsional dan aktor dan keterhubungannya
Sequence	Menunjukkan interaksi yang terjadi antar obyek
Kolaborasi	Diagram interaksi yang menekankan pada organisasi struktur dari obyek-obyek yang mengirim dan menerima pesan
Statechart	Memodelkan perilaku antarmuka, kelas, kolaborasi dan menekankan pada urutan kejadian
Aktivitas	Memodelkan fungsi sistem dan menekankan pada aliran kendali di antara obyek-obyek
Kelas	Menunjukkan sekumpulan kelas, *interface* dan kolaborasi dan keterhubungannya
Obyek	Menunjukkan sekumpulan obyek dan keterhubungannya
Komponen	Menunjukkan organisasi dan kebergantungan di antara sekumpulan komponen
Deployment	Menunjukkan konfigurasi pemrosesan saat jalan dan komponen-komponen yang terdapat di dalamnya

3.1 Diagram *Use-case*



Gambar 2 Diagram *Use-Case* Sistem Interpretasi Alfabet *Pitman Shortand*

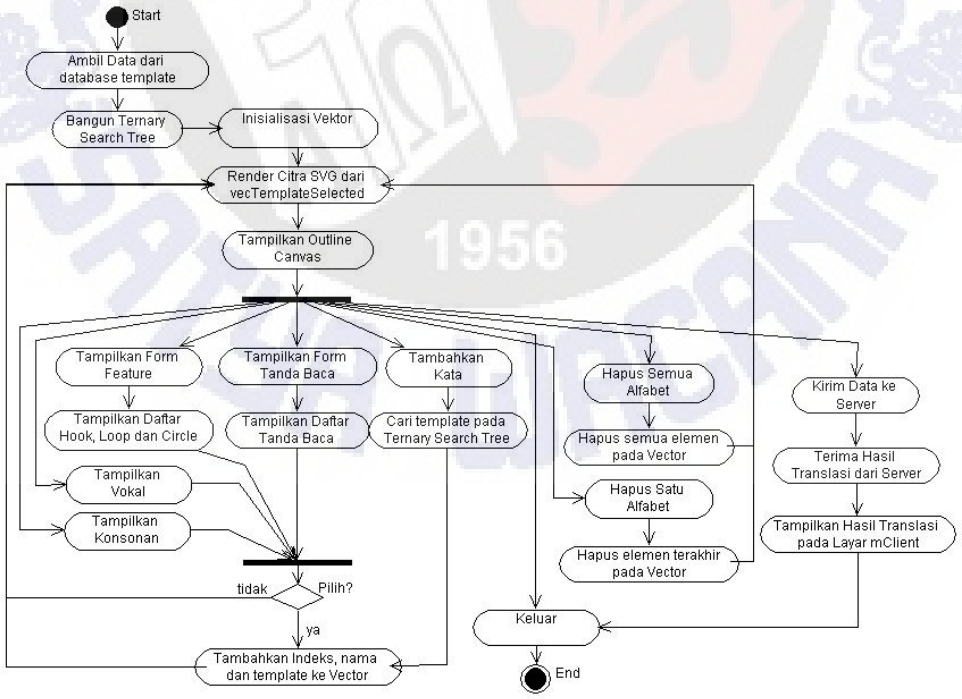
Diagram *use-case* merupakan diagram yang menjelaskan manfaat sistem jika dilihat dari sudut pandang orang atau sesuatu yang berada di luar sistem yang sedang dibangun (aktor). Jenis diagram ini dapat digunakan untuk menangkap *requirements* sistem dan untuk memahami bagaimana sistem seharusnya bekerja [6]. Diagram *use-case* yang digunakan digambarkan pada Gambar 2.

3.2 Diagram Aktivitas

Diagram aktivitas adalah diagram yang memodelkan alur kerja (*workflow*) sebuah proses bisnis dan urutan aktivitas langkah per langkah dalam suatu proses [16]. Perilaku atau proses yang terjadi dalam suatu *use-case* dapat digambarkan melalui diagram ini.

Sistem pada sisi *client* diawali dengan pengambilan data-data *template* yang ada pada basisdata dan menggunakan data-data tersebut untuk membangun *tree*. Data yang ada pada *tree* dapat diambil lagi untuk keperluan penggambaran simbol alfabet. Indeks dari alfabet yang dipilih disimpan pada variabel penampung. Isi dari variabel penampung ini dikirimkan ke *server*.

Data indeks yang diterima dari *client* dibagi-bagi berdasarkan tanda spasi, sehingga diperoleh indeks untuk masing-masing kata. Setiap data indeks dicari pasangan katanya pada kamus yang sudah disediakan. Hasil pencarian digabung lagi dan dikirim kembali ke *server* untuk ditampilkan.

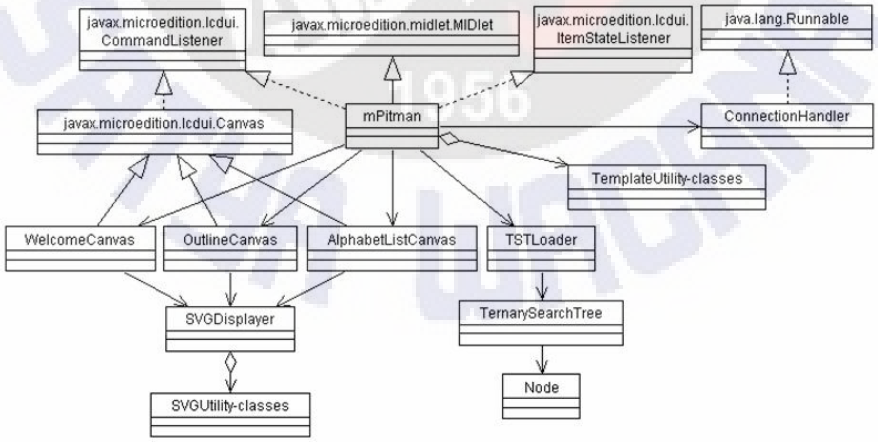


Gambar 3 Diagram Aktivitas Sistem pada *Client Side*

Diagram aktivitas sistem pada sisi *client* dapat digambarkan pada Gambar 3. Sedangkan diagram aktivitas sistem pada sisi *server* dapat digambarkan pada Gambar 4.

Gambar 4 Diagram Aktivitas Sistem pada *Server Side*

3.3 Diagram Kelas

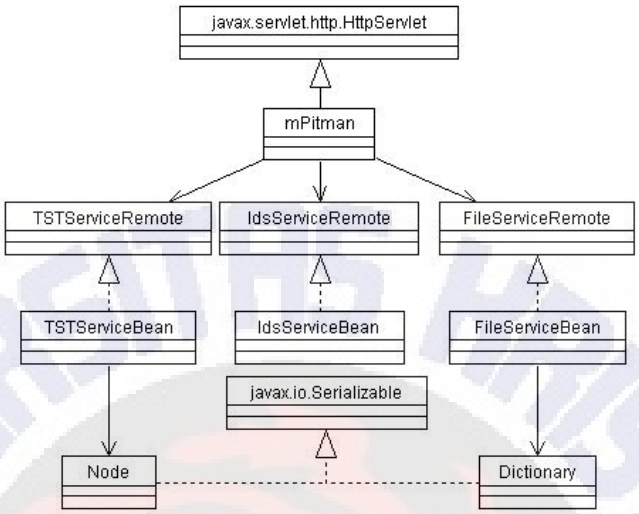


Gambar 5 Diagram Kelas Sistem pada *Client Side*

Diagram kelas merupakan diagram yang membantu dalam visualisasi struktur kelas-kelas dari suatu sistem. Dalam diagram ini, diperlihatkan hubungan antar kelas dan penjelasan detail tiap-tiap kelas [5].

Diagram kelas sistem pada sisi *client* dapat digambarkan pada Gambar

5. Sedangkan diagram kelas sistem pada sisi *server* dapat digambarkan pada Gambar 6.



Gambar 6 Diagram Kelas Sistem pada *Server Side*

4. Hasil dan Pembahasan

Sistem pada sisi *client* dibangun dengan platform *Java Micro Edition* (MIDP 2.0, CLDC 1.1) dan menggunakan paket *optional* JSR-226 (SVG API). Pada sisi *server*, sistem dibangun dengan platform *Java Enterprise Edition* (JEE) dan menggunakan web *container SJSAS Platform Edition* 9.0. Untuk percobaan digunakan *Sun Wireless Toolkit* 2.5.

4.1 Unjuk Kerja Algoritma TST

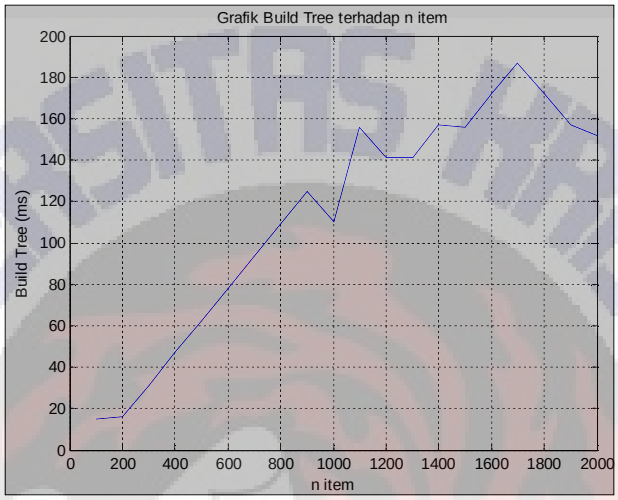
Efisiensi dalam pencarian data pada suatu *tree* dapat diukur melalui notasi big-O. Notasi big-O didefinisikan sebagai suatu ukuran teoritis dari algoritma tertentu, biasanya menyangkut waktu dan memori yang diperlukan, yang diberikan sejumlah (*n* item) masalah. Sehingga, notasi ini dapat digunakan untuk membandingkan efisiensi algoritma yang satu dan lainnya.

Pada *average-case*, *Binary Search Tree* (BST) dapat mencari data dengan kompleksitas waktu yang sebanding dengan tinggi dari *tree* yang terbentuk. Dalam notasi big-O ditulis sebagai $O(^2\log n)$, dimana *n* adalah jumlah *node*. Kondisi ini terpenuhi untuk *balanced-tree*. Pada *worst-case*, *Binary Search Tree* memerlukan waktu sebanding dengan jumlah *node*-nya dan ditulis sebagai $O(n)$. Kasus ini terjadi jika data masukan diurutkan terlebih dahulu.

Pada percobaan, diukur waktu penyusunan *tree* dan waktu pencarian data. Waktu penyusunan *tree* diperoleh dengan menyusun *tree* untuk jumlah item yang berbeda dengan selang yang tetap. Pada masing-masing *tree* yang terbentuk dicari data berdasarkan kata kunci yang tetap.Grafik waktu

penyusunan *tree* terhadap jumlah item ditunjukkan pada Gambar 7.

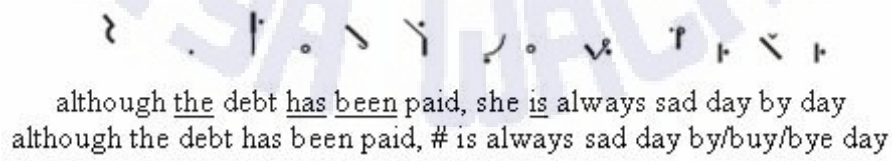
Untuk setiap percobaan, waktu pencarian data pada *tree* selalu bernilai 0 ms. Perbedaan waktu pencarian yang signifikan tidak terlihat karena satuan waktu yang digunakan kurang teliti. Diperlukan satuan waktu yang lebih kecil seperti mikrosekon (us) atau nanosekon (ns). Kendala yang dihadapi adalah tidak tersedianya *method* yang diperlukan pada platform JME untuk mendapatkan satuan waktu yang lebih kecil.



Gambar 7 Grafik Waktu Penyusunan *Tree* Terhadap Jumlah *Item*

4.2 Proses Translasi

Pengujian sistem dilakukan dengan memasukkan input berupa alfabet-alfabet *Pitman Shorthand* yang akan dikirim dan diterjemahkan pada sisi *server*. Hasil terjemahan akan ditampilkan pada sisi *client*. Pengujian dilakukan sebanyak 25 kali dengan input yang berbeda-beda. Contoh ditunjukkan pada Gambar 8.



Gambar 8 Contoh Kalimat dalam Percobaan

Baris pertama merupakan bentuk *shorthand* dari kalimat pertama. *Short-form* atau *grammalogue* digarisbawahi. Sedangkan kalimat kedua merupakan hasil translasi dari *shorthand* pada baris pertama.

Kata-kata yang tidak dapat diterjemahkan disimbolkan dengan tanda # (pagar). Kata yang merupakan homofon ditampilkan dengan kata homofon lainnya yang setara. Pada kalimat di atas (Gambar 8), kata “by” merupakan

homofon, diterjemahkan sebagai “by/buy/bye”.

Hasil percobaan untuk semua kalimat disajikan pada Tabel 2. Simbol **e** mewakili jumlah kata yang tidak dapat diterjemahkan dan simbol **t** mewakili waktu yang diperlukan sistem untuk menerjemahkan kalimat yang diinputkan dalam satuan sekon.

Dari hasil percobaan dapat terlihat bahwa hasil interpretasi belum sempurna karena terdapat beberapa kata yang tidak ditemukan pada berkas kamus di sisi *server*.

Prosentase kegagalan setiap percobaan dihitung dengan rumus berikut:

$$\mathrm{e} = \frac{\textit{Jumlah kata yang salah}}{\textit{Jumlah seluruh kata}} \times 100\% \tag{1}$$

Sedangkan, prosentase kesuksesan setiap percobaan dihitung dengan rumus:

$$s = \frac{\textit{Jumlah kata yang benar}}{\textit{Jumlah seluruh kata}} \times 100\% \tag{2}$$

atau,

$$s = 100\% - \mathrm{e} \tag{3}$$

Berdasarkan ketiga rumus (1), (2) dan (3) tersebut dapat dihitung rerata prosentase keberhasilan ($\hat{s}$) dan kegagalan ($\hat{e}$) untuk semua percobaan. Didapat:

$$\sum_{i=1}^{25} e_i = \left[\frac{1}{13}+\frac{0}{10}+\frac{3}{10}+\frac{3}{10}+\frac{4}{9}+\frac{8}{13}+\frac{5}{17}+\frac{4}{11}+\frac{1}{16}+\frac{3}{11}+\frac{2}{9}+\frac{4}{13}+\frac{4}{13}\right.$$
$$\left.\frac{1}{6}+\frac{2}{8}+\frac{3}{7}+\frac{2}{7}+\frac{2}{11}+\frac{6}{12}+\frac{5}{13}+\frac{2}{7}+\frac{0}{8}+\frac{3}{15}+\frac{3}{9}+\frac{3}{7}\right]$$

$$\hat{e} = \frac{\sum_{i=1}^{25} e_i}{25} = \frac{7{,}01}{25} \text{ x } 100\,\% = 28{,}05\,\% \tag{4}$$

$$\hat{s} = \frac{\sum_{i=1}^{25} s_i}{25} = 100\,\% - \hat{\mathrm{e}} = 100\,\% - 28{,}05\,\% = 71{,}95\,\% \tag{5}$$

Rerata waktu yang diperlukan untuk menerjemahkan setiap kata dapat dihitung dengan rumus berikut :

$$\hat{t} = \frac{\sum_{i=1}^{25} t_i}{\textit{Jumlah seluruh kata yang dipakai}} \tag{6}$$

Tabel 2 Hasil Percobaan Translasi

No	Kalimat Input	t	e
1.	Although the debt has been paid, she is always sad day by day	3,66	1
2.	This is my boat which was purchased 3 days ago	3,42	0
3.	Jane has a family member, Jade, which lives in Japan	3,66	3
4.	I wish, she will come regularly to gain several chances	3,44	3
5.	The flight to Batam is delayed for 30 minutes	3,64	4
6.	Mr. Jake, the principal, do believe in his financial knowledge which is important	3,09	8
7.	One of two kidneys in our body can be taken without affecting the performance of our body	3,23	5
8.	I believe that every man has a different character and ability	2,89	4
9.	They write several long letters in different language to practice and improve their ability in vocabulary	3,53	1
10.	Do you bring the photo that i ordered this morning?	3,69	3
11.	He brings a lot of happiness into her life	4,00	2
12.	He can organise any member of the class to do something for himself	2,94	4
13.	Bob always makes a plan schedule each day and tries to fulfill it	3,00	4
14.	There is an advantage in advertisement	3,77	1
15.	We are having a long holiday this year	3,67	2
16.	She got many opportunities in the past	3,50	3
17.	Most of my friends are having car	3,73	2
18.	My mother is the most famous book writer in this year	3,58	2
19.	I tell the happy news to my dear family in another city	2,95	6
20.	Binary number system consists of two numbers only those are 0 and 1	4,33	5
21.	They will catch him die or alive	3,83	2
22.	Many years had been passed since he came	3,56	0
23.	A detective has to memorize all events deal with the case and analyze all posibilities	3,45	3
24.	That hobo is imbibing his fifth bottle of liquor	3,69	3
25.	Jack will publish his novel this year	3,51	3

$$\sum_{i=1}^{25} t_i = [3{,}66 + 3{,}42 + 3{,}66 + 3{,}44 + 3{,}64 + 3{,}09 + 3{,}23 + 2{,}89 +$$

$$3{,}53 + 3{,}69 + 4{,}00 + 2{,}94 + 3{,}00 + 3{,}77 + 3{,}67 + 3{,}50 +$$

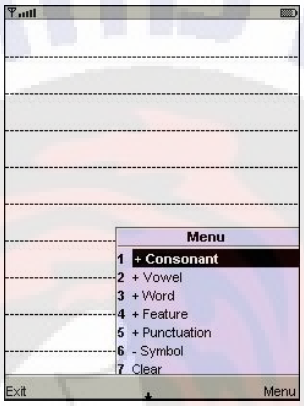
$$3{,}73 + 3{,}58 + 2{,}95 + 4{,}33 + 3{,}83 + 3{,}56 + 3{,}45 + 3{,}69 + 3{,}51]$$

$$= 87{,}76 \textit{ sekon}$$

$$\hat{t} = \frac{87{,}76}{265} = 0{,}33 \; sekon$$

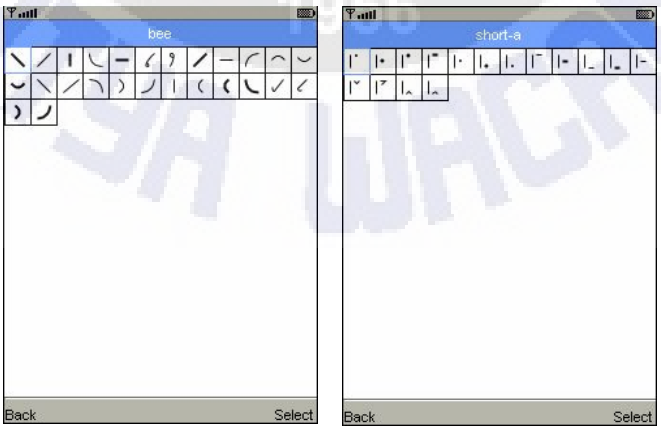
Dari percobaan tersebut, diperoleh rerata prosentase keberhasilan sebesar 71,95%, rerata prosentase kegagalan sebesar 28,05% dan rerata waktu proses yang diperlukan untuk menerjemahkan satu kata adalah 0,33 sekon.

Berikut ini adalah gambar-gambar *software* yang telah dibuat untuk mengaplikasikan teori-teori yang telah dibahas pada tulisan ini. Gambar 9 menunjukkan tampilan program pada saat pertama kali dijalankan. Bagian tersebut dinamakan kanvas *outline.*



Gambar 9 Kanvas Outline

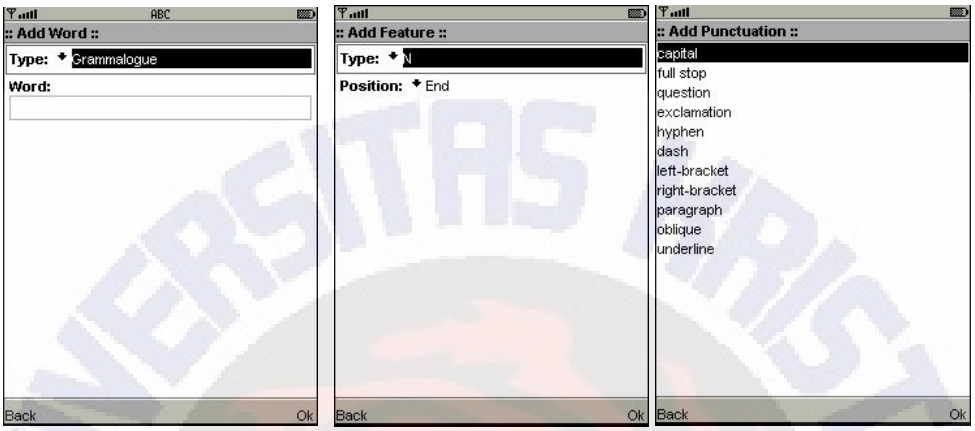
Gambar 10 terdiri dari dua bagian, bagian (a) menunjukkan daftar alfabet *Pitman Shorthand* untuk simbol konsonan yang dapat dipergunakan dalam program, sedangkan bagian (b) menunjukkan daftar alfabet *Pitman Shorthand* untuk simbol vokal yang dapat dipergunakan dalam program.



(a) (b)

Gambar 10 Alfabet *Pitman Shorthand* yang dapat Dipergunakan dalam Program, (a) Simbol Konsonan dan (b) Simbol Vokal

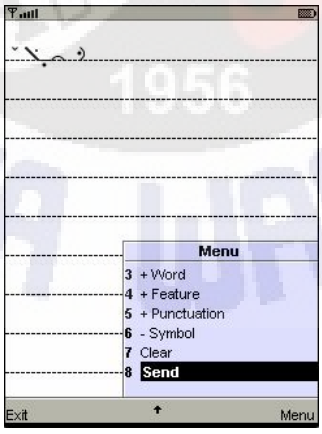
Gambar 11 terdiri dari tiga bagian, bagian (a) menunjukkan tampilan untuk fungsi *Add Word* atau menambah kata, bagian (b) menunjukkan tampilan untuk fungsi *Add Feature* atau menambah fitur, dan bagian (c) menunjukkan tampilan untuk *Add Punctuation* atau menambah tanda baca.



(a) (b) (c)

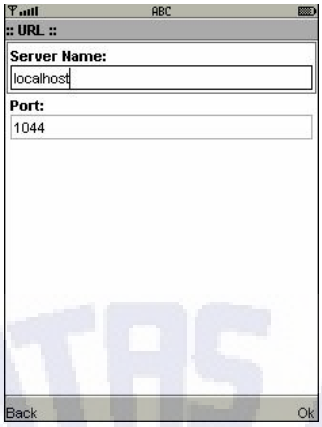
Gambar 11 Fungsi-fungsi dalam Program yaitu (a) *Add Word,* (b) *Add Feature,* dan (c) *Add Punctuation*

Gambar 12 memperlihatkan tampilan *outline* dari kata-kata yang telah diterjemahkan oleh program. Sedangkan Gambar 13 memperlihatkan tampilan form URL untuk me-*load* atau mengirimkan data hasil terjemahan.

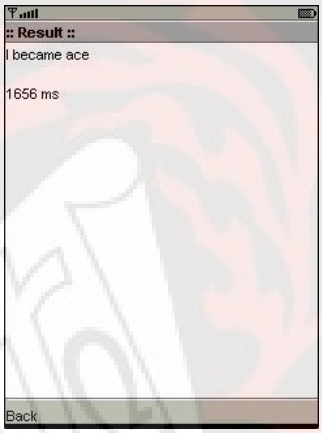


Gambar 12 *Outline* yang Diterjemahkan

Gambar 14 memperlihatkan tampilan hasil interpretasi dari proes penerjemahan yang dilakukan oleh program.



Gambar 13 Form URL



Gambar 14 Hasil Interpretasi

5. Simpulan

Sistem mampu menerjemahkan masukan yang berupa alfabet-alfabet *Pitman Shorthand* menjadi teks dalam bahasa Inggris. Pada percobaan yang dilakukan sebanyak 25 kali dengan kalimat yang berbeda-beda, diperoleh prosentase keberhasilan sebesar 71,95% dan rerata waktu yang diperlukan 0,33 sekon per kata. Waktu yang diperlukan untuk menyusun *tree* semakin besar dengan bertambahnya jumlah item yang digunakan. Sedangkan, waktu pencarian data pada *tree* tidak menunjukkan perbedaan pada setiap percobaan, yaitu 0 ms.Kegagalan dalam proses translasi bergantung pada ketersediaan data dalam kamus. Semakin lengkap data pada kamus, prosentase keberhasilan semakin tinggi. Untuk kepentingan manajemen data kamus disediakan aplikasi *desktop* yang dibangun dengan *platform Java Standard Edition* (JSE).

Beberapa saran pengembangan yang dapat dilakukan antara lain: (1) Dapat menggunakan *TinyLine SVG API* sebagai alternatif untuk me-*render* alfabet; (2) Sistem pada sisi *client* dapat diimplementasikan menggunakan

Symbian C++. Sedangkan, sistem pada sisi server dapat digantikan dengan platform *.NET*; dan (3) Konsep *web service* dapat diterapkan untuk proses komunikasi antara *client* dan *server*.

6. Daftar Pustaka

[1] Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki.

[2] Anonym, 2000, *Shorthand*, In BBC Homepage: http://www.bbc.co.uk/dna/h2g2/A291250. Retrieved January 15, 2007.

[3] Eisenberg, J. David, 2001, *An Introduction to Scalable Vector Graphics*, O'REILLY xml.com. from: http://www.xml.com/pub/a/2001/03/21/svg.html. Retrieved January 12, 2007.

[4] Dobbs, 2001, Ternary Search Trees, Dr. Dobbs Portal: http://www.ddj.com/dept/windows/ 184410528. Retrieved January 12, 2007.

[5] Hariyanto, Bambang, 2004, *Rekayasa Sistem Berorientasi Objek*, Penerbit INFORMATIKA, Bandung.

[6] Nugroho, Adi, 2005, *Rational Rose untuk Pemodelan Berorientasi Objek*". Penerbit INFORMATIKA, Bandung.