Jurnal Teknologi Maklumat & Multimedia **8**(2010): 75 - 87

Pengecaman Elemen Asas Pengaturcaraan Berorientasikan Objek Secara Pemprosesan Bahasa Tabii

NOMARIANI A. RAZIK & NAZLIA OMAR

ABSTRAK

Kebanyakan pelajar terutamanya pelajar tahun satu di pusat pengajian tinggi menghadapi masalah ketika memulakan penulisan aturcara terutamanya setelah mendapat soalan pengaturcaraan berorientasikan objek. Jadi, sebuah prototaip iaitu OOP Tool yang menggunakan set peraturan heuristik telah dibangunkan untuk menentukan elemen asas pengaturcaraan berorientasikan objek berasaskan dokumen yang menggunakan teknik Pemprosesan Bahasa Tabii. Penentuan elemen asas pengaturcaraan berorientasikan objek ini menggunakan set peraturan heuristik untuk penentuan ketepatan di mana satu set peraturan heuristik telah dibangunkan dengan menggunakan metodologi yang berasaskan satu set data latihan. Hasil daripada kajian ini, satu set peraturan heuristik dan satu prototaip yang dinamakan OOP Tool telah dibangunkan untuk menentukan elemen asas pengaturcaraan berorientasikan objek seperti kelas, pembolehubah/atribut dan fungsi/metod. Penilaian bagi OOP Tool telah dibuat menggunakan satu set data ujian di mana keputusan yang diperolehi adalah 90.5% bagi dapatan semula, 76.3% bagi kejituan, 37.7% bagi lebihan-jana (over-generated) dan 9.5% bagi kurangan-jana (under-generated). OOP Tool sangat berguna kepada pelajar tahun satu di pusat pengajian tinggi dalam mempelajari pengaturcaraan berorientasikan objek.

Kata kunci: Pengaturcaraan Berorientasikan Objek, Pemprosesan Bahasa Tabii, Heuristik.

ABSTRACT

Most of the students especially first year students in universities or colleges face problems in writing programs when given object oriented programming problems. Thus, a prototype called OOP Tool which utilizes heuristics was developed to determine the basic elements of object oriented programming based on documented programming problems using natural language processing techniques. The determination of the elements was based on a set of heuristics to determine the accuracy of the results. The heuristics were developed using a methodology based on a training set. From this research, a set of heuristic was developed which is capable of determining the basic elements of object oriented programing such as classes, variables/ attributes and functions/methods. These heuristics were realized in terms of implementation in a prototype called OOP Tool. To further support the contribution, an evaluation of OOP Tool is carried out. The results from the test set showed that the prototype achieved 90.5% recall and 76.3% precision, 37.7% over-generated and 9.5% under-generated. This OOP Tool can serve as a useful tool for first year's students in university or college in learning object-oriented programming.

Keywords: Object Oriented Programming, Natural Language Processing, Heuristic.

PENGENALAN

Pengaturcaraan berorientasikan objek ialah sebuah atur cara komputer yang terdiri daripada sekumpulan unit atau objek. Untuk membolehkan perkomputeran berlaku, setiap objek berupaya untuk menerima dan menghantar mesej kepada objek lain. Dengan cara ini, mesej dapat ditangani oleh sebahagian daripada kod, tetapi boleh juga ditangani oleh blok kod yang lain secara lancar. Kölling (1999) menyatakan bahawa dalam beberapa tahun kebelakangan ini, pengaturcaraan berorientasikan objek menjadi paradigma pengaturcaraan yang paling berpengaruh. Ia digunakan secara meluas dalam sektor akademik dan industri dan terdapat kurikulum ini hampir di semua universiti dan kolej.

Walau bagaimanapun, para pelajar menghadapi kesukaran untuk mengaplikasikan konsep berorientasikan objek ini terutamanya untuk memulakan proses pengaturcaraan. Menurut Harmain & Gaizauskas (2003), sejak kebelakangan ini, teknologi berorientasikan objek telah digunakan bermula daripada fasa pengaturcaraan pembangunan perisian untuk fasa awal Analisis dan Rekakabentuk. Analisis berorientasikan objek adalah antara fasa yang paling susah dan kritikal kerana input bagi proses ini diberikan dalam bahasa tabii (contohnya Bahasa Inggeris).

Kaedah baru yang kreatif perlu disediakan bagi membantu pelajar mengatasi masalah tersebut. Kaedah yang dicadangkan adalah dengan mewujudkan alatan yang dapat menyatakan atau menentukan elemen asas di dalam pengaturcaraan berorientasikan objek dengan hanya memasukkan soalan pengaturcaraan ke persekitaran alatan tersebut. Dengan ini, pelajar mendapat gambaran awal sebelum memulakan proses pengaturcaraan.

PENGATURCARAAN BERORIENTASIKAN OBJEK

Sebelum memulakan sesuatu penulisan pengaturcaraan berorientasikan objek, seseorang pelajar itu hendaklah memahami maklumat yang terkandung di dalam soalan pengaturcaraan. Berdasarkan soalan pengaturcaraan yang diberi, pelajar perlu menentukan elemen asas yang diperlukan untuk memulakan proses penulisan pengaturcaraan berorientasikan objek ini. Sebenarnya terdapat banyak elemen yang perlu dipertimbangkan dalam penulisan aturcara berorientasikan objek tetapi di dalam kajian ini hanya tiga elemen sahaja yang akan diperbincangkan iaitu nama kelas, pembolehubah/atribut dan fungsi/metod. Kelas mentakrifkan atribut dan fungsi untuk semua objek daripada jenis tertentu. Booch (1994) menyatakan kelas adalah kumpulan objek yang berkongsi struktur dan perlakuan yang sama.

PENGGUNAAN PEMPROSESAN BAHASA TABII DI DALAM ANALISIS BERORIENTASIKAN OBJEK

Banyak kajian telah dilakukan ke atas penggunaan bahasa tabii dalam melakukan analisis berorientasikan objek. Dalam analisis berorientasikan objek, elemen asasnya perlu ditentukan terlebih dahulu untuk memastikan proses pengaturcaraan berjalan dengan lancar.

Abbot (1983) menyatakan melalui perkataan dan frasa dalam Bahasa Inggeris, jenis data, objek, operator dan struktur kawalan boleh ditentukan. Kata nama dan frasa kata nama di dalam strategi tidak formal merupakan petunjuk yang baik bagi mengenalpasti objek serta sifatnya. Semua kata nama am di dalam teks yang ditulis di dalam Bahasa Inggeris , contohnya "*epal*" dikenalpasti sebagai kelas. Frasa kata nama pula, boleh digunakan untuk merujuk kepada objek yang lebih spesifik.

CM-Builder (Harmain & Gaizauskas 2003) adalah perisian berasaskan peralatan CASE di mana matlamatnya adalah menyokong peringkat analisis bagi pembangunan perisian dalam rangka kerja berorientasikan objek. Ia menggunakan teknik pemprosesan bahasa tabii untuk menganalisis dokumen keperluan sistem yang ditulis di dalam Bahasa Inggeris dan menghasilkan permulaan Model Konseptual Berorientasikan Objek yang diwakilkan di dalam UML (*Unified Modelling Language*). Hasil daripada analisis

itu dapat dikenalpasti kelas-kelas objek, atribut dan hubungan antaranya. Antara peraturan yang dihasilkan adalah semua kata nama yang berada dalam teks dikenalpasti sebagai kelas, kata sifat dikenalpasti sebagai atribut dan semua kata kerja (*verb*) pula dikenalpasti sebagai hubungan.

LIDA (*Linguistic Assistant for Domain Analysis*) (Overmyer et al. 2001) merupakan metodologi yang dapat membantu penganalisis membangunkan model-model berorientasikan objek. Metodologi LIDA ini dimulakan dengan diskripsi teks bagi konsep operasi sistem yang dicadangkan. Penganalisis akan membaca dan menanda kata nama dan kata kerja dan selepas itu cuba untuk mengenalpasti objek dan fungsi/metod yang bersesuaian. Dengan menggunakan kelas perkataan, objek dan fungsi/metod dikenalpasti di mana kata nama adalah untuk kelas, kata kerja untuk hubungan dan kata sifat untuk atribut. Kebanyakan teks dokumen yang menerangkan fungsi sistem mengandungi banyak kata nama, kata kerja, kata penerang dan kata sifat di mana ia adalah amat berguna dalam usaha mengenalpasti objek.

GOOAL (*Graphic Object Oriented Analysis Laboratory*) (Gonzalez & Kalita 2002) adalah metodologi yang menerima gambaran masalah dalam bahasa tabii dan menghasilkan model objek. Atribut, kelakuan/ sifat dan hubungan adalah merupakan komponen yang entiti (yang diwakilkan oleh kata nama). Ketiga-tiga komponen ini boleh diperolehi melalui analisis peranan setiap kata nama yang terdapat di dalam teks permintaan dan kata kerja yang mentakrifkan peranan ini. Mereke mendapati setiap kata nama adalah sebagai kelas.

Liu et al. (2004) pula menyatakan bahawa kelas-kelas objek dan ciri-cirinya boleh dikenalpasti daripada permintaan yang dinyatakan dalam bahasa tabii dan model kelas dihasilkan berdasarkannya. Mereka telah membangunkan satu kaedah untuk generasi model *use case*, penentuan objek dan memodelkan kelas berdasarkan permintaan di dalam bahasa tabii. Kaedah ini adalah berasaskan *Rational Unified Process* (RUP). Bagi melaksanakan kaedah itu, mereka membangunkan satu alatan yang diberi nama UCDA (*Use-Case driven Development Assistant*). Antara ciri-ciri UCDA adalah menghuraikan keperluan-keperluan yang dinyatakan di dalam bahasa tabii dan mengenalpasti pelaku dan *use cases* serta menghasilkan gambar rajah *use-case*, membantu pengguna menamatkan spesifikasi *use-case*, mengenalpasti kelas dan menghasilkan gambar rajah yang baik, mengsahihkan analisis model kelas melalui gambar rajah yang baik dan menghasilkan model kelas.

Requirements Elicitor (Mala & Uma 2006) merupakan satu sistem yang menerima input kenyataan masalah yang kenyataan tersebut kemudiannya dipecahkan kepada ayat demi ayat oleh pemecah ayat untuk perlabelan ayat. Setiap ayat dikenalpasti untuk dilabelkan supaya kelas perkataan untuk setiap perkataan diperolehi. Kemudian pemetaan NL-OOML (*Natural Language-Object Oriented Modelling Language*) menerima input deskripsi masalah yang telah dibersihkan. Terdapat beberapa peraturan berasaskan pendekatan telah dibina untuk mengenalpasti elemen berorientasikan objek. Antaranya adalah kata nama yang tidak mempunyai atribut tidak akan dipilih menjadi kelas, jika terdapat dua kata nama dalam satu ayat maka kata nama yang pertama akan dipilih menjadi kelas manakala kata nama yang kedua menjadi atribut dan kata kerja pula akan menjadi fungsi/metod.

Berdasarkan kajian lepas, didapati kebanyakannya adalah berkaitan dengan analisis berorientasikan objek. Walaupun analisis berorientasikan objek adalah antara segmen yang penting di dalam bidang pembangunan sistem dan juga dalam pengajaran dan pembelajaran tetapi pengaturcaraan berorientasikan objek juga tidak kurang pentingnya. Kajian yang berkaitan dengan penentuan elemen asas bagi pengaturcaraan berorientasikan objek adalah kurang. Jadi, untuk memenuhi keperluan ini maka kajian ini dijalankan.

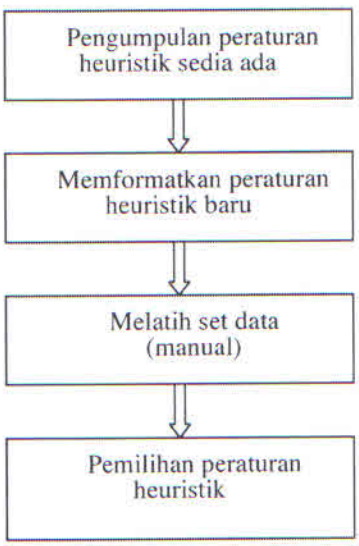
METODOLOGI PEMBANGUNAN PERATURAN HEURISTIK

Heuristik adalah gerak hati, pertimbangan, ilham, petua yang terbina daripada pengalaman. Heuristik merupakan panduan untuk menyelesaikan masalah atau membuat keputusan dengan lebih cepat berbanding kaedah rambang. Perkataan '*heuristic*' berasal daripada perkataan Greek iaitu "heuriskein' yang bermaksud

'untuk menemui'. Menurut Kamus Dewan Bahasa dan Pustaka (2005), heuristik membolehkan atau dapat membantu seseorang mempelajari, mengetahui dan sebagainya sesuatu perkara melalui pengalaman sendiri (bukan kaedah pengajaran atau pembelajaran). Heuristik memberikan gambaran maklumat yang berasaskan pengalaman.

Tujuan pembinaan peraturan heuristik ini ialah untuk mendapatkan peraturan heuristik yang akan digunakan semasa proses pembinaan prototaip. Peraturan heuristik ini digunakan untuk menentukan elemen pengaturcaraan berorientasikan objek seperti kelas objek dan ahlinya. Rajah 1 menunjukkan langkah yang dilakukan untuk mendapatkan peraturan heuristik yang diperlukan.

Pada awal kajian, pengumpulan peraturan heuristik sedia ada merupakan proses mendapatkan heuristik yang telah dibina oleh pengkaji yang lain melalui pembacaan kesusasteraan. Tumpuan kajian pada peringkat ini adalah kepada penerbitan heuristik yang lepas yang berasaskan kepada sintaks bahasa tabii untuk menentukan elemen berorientasikan objek daripada spesifikasi bahasa tabii. Berdasarkan peraturan heuristik yang diperolehi itu, set keputusan data latihan daripada soalan-soalan pengaturcaraan akan diperolehi. Tujuannya adalah untuk memformatkan peraturan heuristik baru. Selepas itu, set data tersebut akan dilatih untuk mendapatkan peraturan heuristik baru. Setelah mendapat peraturan heuristik baru maka proses pemilihan peraturan heuristik iaitu gabungan antara heuristik sedia ada dengan heuristik baru dilakukan. Tujuannya adalah hanya heuristik yang sesuai sahaja dipilih dan digunakan semasa proses pembinaan prototaip.



RAJAH 1. Langkah untuk mendapat peraturan heuristik

HEURISTIK UNTUK MENENTUKAN ELEMEN ASAS PENGATURCARAAN BERORIENTASIKAN OBJEK

Peraturan heuristik boleh digunakan sebagai garis panduan kepada yang tidak berpengalaman untuk melakukan sesuatu yang telahpun digunakan atau dilaksanakan oleh yang berpengalaman. Dalam konteks analisis berorientasikan objek, reka bentuk peraturan heuristik menggambarkan reka bentuk pengalaman daripada pembangun berorientasikan objek

PERATURAN HEURISTIK SEDIA ADA

Peraturan heuristik yang terhasil daripada kajian yang lepas (Harmain & Gaizauskas 2003; Overmyer et al. 2001; Mala & Uma 2006; Abbot 1983; Nanduri & Rugaber 1995) untuk menentukan elemen berorientasikan

objek daripada spesifikasi keperluan bahasa tabii akan diterangkan secara terperinci di sini. Senarai dalam Jadual 1 menunjukkan singkatan nama yang digunakan dalam menentukan kategori bagi setiap heuristik.

JADUAL 1. Singkatan nama yang digunakan untuk kategori peraturan heuristik

Singkatan nama	Kategori peraturan heuristik
HBK	Bukan kelas
HBF	Bukan fungsi
HK	Jenis kelas
HP	Jenis pembolehubah/atribut
HF	Jenis Fungsi/metod

PERATURAN HEURISTIK UNTUK MENENTUKAN KELAS

1. Heuristik HK1 : Semua kata nama di dalam teks diambil sebagai kelas (Harmain & Gaizauskas 2003; Overmyer et al. 2001).
2. Heuristik HK2 : Apabila dua kata nama wujud dalam satu ayat, kata nama yang pertama diambil sebagai kelas (Mala & Uma 2006).

   Contoh: "*a rectangle's length*" di mana kata nama "*rectangle*" akan diambil sebagai kelas.
3. Heuristik HK3 : Kata nama yang tidak mempunyai atribut tidak akan dipilih menjadi kelas (Mala & Uma 2006).

   Contoh: "*a program need an input*" di mana kata nama "*program*" merupakan kata nama yang tidak mempunyai atribut.
4. Heuristik HK4 : Kata nama am diambil sebagai kelas (Abbot 1983).

   Contoh: "*an apple*", "*a table*", "*an idea*" "*a telephone*", "*a terminal*" dan lain-lain lagi.

PERATURAN HEURISTIK UNTUK MENENTUKAN PEMBOLEHUBAH/ATRIBUT

1. Heuristik HP1 : Apabila dua kata nama wujud dalam satu ayat, kata nama yang kedua diambil sebagai pembolehubah/atribut (Mala & Uma 2006).

   Contoh: "*a rectangle's length*" di mana kata nama "*length*" akan diambil sebagai

   Pembolehubah

PERATURAN HEURISTIK UNTUK MENENTUKAN FUNGSI/METOD

1. Heuristik HF1 : Kata kerja bagi kata nama am diambil sebagai fungsi (Mala & Uma 2006; Nanduri & Rugaber 1995).

   Contoh: "*write a program*" di mana kata kerja "*write*" akan diambil sebagai fungsi.
2. Heuristik HF2 : Kata sifat digunakan untuk menentukan fungsi (Nanduri & Rugaber 1995)

CADANGAN PERATURAN HEURISTIK BARU

Berdasarkan keperluan untuk mempertingkatkan ketepatan keputusan maka satu set tambahan peraturan heuristik baru diperkenalkan. Peraturan heuristik ini boleh digambarkan sebagai peraturan heuristik sintaktik tambahan yang mana boleh membantu dalam mempertingkatkan lagi keputusan dalam penentuan elemen-elemen asas pengaturcaraan berorientasikan objek daripada spesifikasi keperluan bahasa tabii iaitu Bahasa Inggeris. Hanya peraturan heuristik yang berasaskan kepada sintaks sahaja dipertimbangkan dan dicadangkan.

PERATURAN HEURISTIK BARU UNTUK MENENTUKAN KELAS

1. Heuristik HK5 : Sesuatu kata nama yang sebelumnya terdapat perkataan "class" akan diambil sebagai kelas

   Contoh: "define a class addressType that can store a street address, city, state and ZIP code" di mana kata nama "addressType" akan digunakan sebagai kelas.

PERATURAN HEURISTIK BARU UNTUK MENENTUKAN PEMBOLEHUBAH/ATRIBUT

1. Heuristik HP3 : Nombor kardinal yang wujud di belakang sesuatu abjad atau kata nama di dalam teks boleh diambil dan digunakan sebagai pembolehubah.

   Contoh: "value of num1 and num2" di mana "num1" dan "num2" boleh diambil sebagai pembolehubah.
2. Heuristik HP4 : Simbol yang wujud di dalam teks boleh diambil sebagai pembolehubah.

   Contoh: "write the definition of the function nextChar that sets the value of z to the next character stored in z" di mana 'z' merupakan simbol yang memegang sesuatu nilai maka ia boleh digunakan sebagai pembolehubah.
3. Heuristik HP5 : Sesuatu kata nama yang sebelumnya terdapat nombor kardinal akan diambil atau dikenalpasti sebagai pembolehubah.

   Contoh: "write a program that asks the user to enter two numbers" di mana perkataan "numbers" akan diambil sebagai pembolehubah.
4. Heuristik HP6 : Sesuatu kata nama yang sebelumnya terdapat kata kerja 'read, reads, print, prints, display' atau kata nama 'input, inputs' akan diambil sebagai pembolehubah.

   Contoh: "Find and prints sum of the numbers" di mana "sum" akan diambil sebagai pembolehubah.

PERATURAN HEURISTIK BARU UNTUK MENENTUKAN FUNGSI/METOD

1. Heuristik HF3 : Sesuatu kata nama yang sebelumnya terdapat perkataan "*function*" atau "*method*" akan diambil atau dikenalpasti sebagai fungsi/metod.

   Contoh: "*a function calculateGrate to determine and return each student's grate*" di mana perkataan "*getNumber*" akan diambil sebagai fungsi.
2. Heuristik HF4 : Kata nama, kata kerja atau kata sifat yang selepasnya terdapat buka kurungan dan tutup kurungan akan diambil sebagai fungsi/metod.

   Contoh: "*Write a function named liquid ( ) that has an integer number parameter*" di mana perkataan "*liquid*" akan diambil sebagai fungsi.

PERATURAN HEURISTIK UNTUK MENENTUKAN BUKAN KELAS

1. Heuristik HBK : Jika terdapat kata nama yang tergolong dalam set X di mana X = {*program, prompts, user, function, class, statement,input*}, kata nama tersebut dikecualikan daripada menjadi kelas.

Kata nama seperti *program, prompts, user, function, class, statement,input* dan *define* tidak sesuai digunakan sebagai kelas. Sebagai contoh, kata nama *program* adalah merujuk kepada aturcara yang ditulis dan ia tidak sepatutnya digunakan sebagai kelas. Sebagai contoh ayat:

"*write a program that prompts the user to input the length and width*".

"*Write C++ statement that multiplies a value of num1 by 2*".

"*Define a class addressType that can store a street address, city, state and Zip code*".

PERATURAN HEURISTIK UNTUK MENENTUKAN BUKAN FUNGSI

1. Heuristik HBF : Jika terdapat kata kerja yang tergolong dalam set Y di mana Y = {*write, find, finds, assume, use, define*}, kata kerja tersebut dikecualikan daripada menjadi fungsi.

Terdapat beberapa kata kerja yang selalu wujud di dalam soalan pengaturcaraan. Antaranya adalah *write, finds, assume, use, define* dan *prompt* dan kata kerja tersebut tidah sesuai digunakan sebagai fungsi. Sebagai contoh ayat:

"*Assume that the maximum number of students*"

"*Write a program that reads student's name*"

SET DATA LATIHAN

Untuk membina peraturan heuristik yang baru, satu ujian secara manual dilakukan sebelum perlaksanaan prototaip OOP Tool. Peringkat ini merupakan fasa yang penting untuk memastikan peraturan-peraturan heuristik yang digunakan adalah berkesan dan memberikan hasil yang baik sebelum fasa perlaksanaan bermula. 23 contoh soalan pengaturcaraan yang ditulis dalam Bahasa Inggeris telah dipilih untuk set latihan di mana contoh soalan ini dipilih daripada buku teks pengaturcaraan.

KEPUTUSAN SET DATA LATIHAN

Jadual 2 menunjukkan sumbangan yang dihasilkan oleh peraturan heuristik sedia ada dan heuristik baru. Berdasarkan kepada jadual tersebut, kebanyakan peraturan heuristik sedia ada telah digunakan di dalam set latihan tetapi ia menyumbangkan lebih banyak keputusan salah dengan 82% salah dan hanya 18% betul. Sebab utama keputusan salah lebih tinggi daripada keputusan betul adalah kerana kebanyakan peraturan heuristik sedia ada tidak memenuhi spesifikasi keperluan yang tepat bagi menentukan elemen asas pengaturcaraan berorientasikan objek. Sebagai contoh, heuristik HK1 dan heuristik HK4 merupakan penyumbang bilangan salah yang terbesar. Ini adalah kerana kedua-dua heuristik sedia ada ini terlalu umum, menyebabkan skop penggunaannya terlalu luas dan hampir setiap perkataan di dalam set data latihan boleh menggunakan heuristik ini. Walau bagaimanapun, heuristik baru telah menymbangkan keputusan yang lebih baik dengan 88% betul dan 12% salah. Ini adalah kerana heuristik baru menepati atau

memenuhi struktur sintaktik ayat dan kebanyakan perkataan atau frasa bagi menjana elemen pengaturcaraan berorientasikan objek.

Berdasarkan heuristik sedia ada dan yang baru, pemilihan peraturan heuristik perlu dilakukan untuk digunakan semasa pembangunan prototaip. Berdasarkan keputusan set data latihan, terdapat beberapa heuristik sedia ada yang menghasilkan keputusan yang tidak baik, tidak dipilih untuk proses pembangunan prototaip.

JADUAL 2. Sumbangan peraturan heuristik sedia ada dan peraturan heuristik baru

Peraturan heuristik sedia ada				Peraturan heuristik baru			
Heuristik	Bil. Betul	Bil. Salah	Jum Bil	Heuristik	Bil. Betul	Bil. Salah	Jum Bil
HK1	20	167	187	HBF	28	0	28
HK2	4	21	25	HBK	51	3	54
HK3	0	0	0	HK5	2	2	4
HK4	20	167	187	HP3	2	0	2
HP1	26	10	36	HP4	2	0	2
HP2	24	14	38	HP5	8	3	11
HF1	8	57	65	HP6	9	7	16
HF2	2	31	33	HF3	2	0	2
				HF4	5	0	5
JUMLAH	104	467	571	JUMLAH	109	15	124
% JUMLAH	18	82		% JUMLAH	88	12	

SENI BINA OOP TOOL

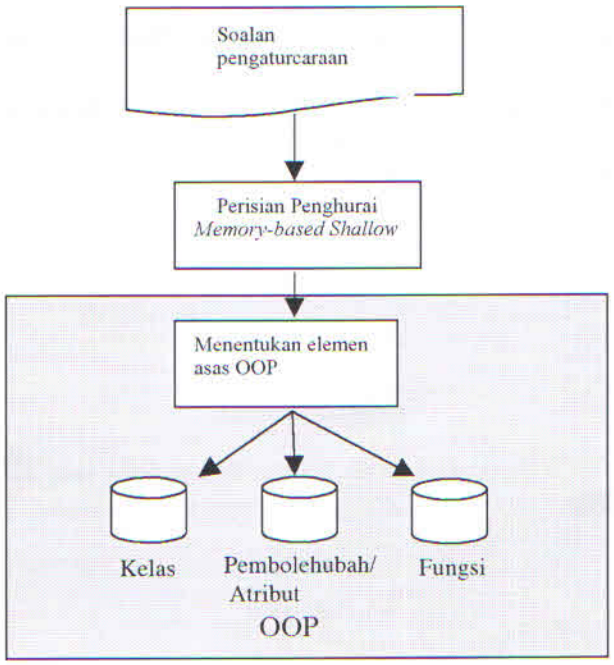
Rajah 2 menunjukkan seni bina bagi OOP Tool. OOP Tool telah dibangunkan menggunakan bahasa pengaturcaan Perl. Penglibatan pemprosesan bahasa tabii di dalam proses ini adalah semasa menterjemahkan soalan pengaturcaraan kepada elemen asas pengaturcaraan berorientasikan objek.

Proses dimulakan dengan membaca teks input yang terdiri daripada soalan pengaturcaraan dalam Bahasa Inggeris. Selepas itu, penghurai diperlukan untuk

$$Precision = \frac{N_{betul}}{N_{betul} + N_{salah}}$$

melabelkan ayat Bahasa Inggeris itu kepada penglabel kelas perkataan (*Part of Speech – POS tagger*) sebelum proses seterusnya dilakukan. Penghurai yang digunakan adalah *Memory-Based Shallow Parser* (MBSP). Teks yang telah dilabelkan kemudiannya dimasukkan ke dalam persekitaran OOP Tool untuk menentukan elemen asas pengaturcaraan berorientasikan objek. Berikut merupakan langkah yang perlu dilakukan untuk menentukan elemen asas pengaturcaraan berorientasikan objek daripada soalan pengaturcaraan.

- Langkah 1 : Melabelkan kelas-kelas perkataan (*Part of Speech tagging*) dengan menggunakan *Memory-Based Shallow Parser*
- Langkah 2 : Membaca input bahasa tabii ke dalam sistem
- Langkah 3 : Menggunakan atau melaksanakan heuristik
- Langkah 4 : Penentuan elemen asas pengaturcaraan berorientasikan objek
- Langkah 5 : Paparan hasil



RAJAH 2. Senibina bagi OOP Tool

PENILAIAN

Kaedah ukuran penilaian yang digunakan dalam kajian ini adalah berdasarkan kaedah penilaian yang digunakan untuk *Information Extraction system* (Grishman dan Sundheim 1996) iaitu dapatan semula dan kejituan. Nilai kedua-dua ukuran ini hendaklah seboleh mungkin menghampiri 100% bagi mana-mana sistem yang digunakan.

Definisi dapatan semula dan kejituan yang diterangkan dalam kajian ini adalah sama seperti mana yang digunakan dalam CM-Builder (Harmain dan Gaizauskas 2003). Selain daripada ukuran dapatan semula dan kejituan, terdapat dua lagi ukuran digunakan iaitu lebihan-jana dan kurangan-jana (Omar et al. 2004). Kedua-dua ukuran ini hendaklah seboleh mungkin menghampiri 0%.

DAPATAN SEMULA

Dapatan semula adalah peratus ketepatan yang wujud dalam maklumat yang dihasilkan oleh sistem. Maklumat yang tepat tersebut kemudiannya dibandingkan dengan keputusan yang dihasilkan oleh penganalisis pakar atau *jawapan sebenar*. Berikut merupakan formula yang digunakan untuk mengira dapatan semula:

$$Dapatan\ semula = \frac{N_{betul}}{N_{jawapan\_sebenar}}$$

Berdasarkan kepada formula di atas, bilangan betul yang dihasilkan oleh OOP Tool diwakilkan dengan N_{betul} dan $N_{jawapan\_sebenar}$ pula adalah merujuk kepada bilangan yang dihasilkan oleh penganalisis atau *jawapan sebenar.*

KEJITUAN

Kejituan adalah peratus ketepatan maklumat yang telah dihasilkan. Kejituan menggambarkan ketepatan sistem di dalam mendapatkan keputusan yang tepat. Berikut merupakan formula standard yang digunakan untuk mengira kejituan:

Dalam kajian ini, formula: $Kejituan = \frac{N_{betul}}{N_{betul} + N_{salah}}$ yang digunakan untuk mengira kejituan adalah sama seperti formula di atas tetapi N_{salah} yang dinyatakan di dalam formula di atas adalah mewakili bilangan salah yang telah dihasilkan oleh sistem. Di dalam kajian ini, N_{salah} telah diubahsuai kepada $N_{lebihan\text{-}jana}$ di mana $N_{lebihan\text{-}jana}$ mewakilkan bilangan lebihan-jana yang dihasilkan oleh OOP Tool. Oleh itu, formula yang digunakan adalah seperti yang ditunjukkan di bawah:

$$Kejituan = \frac{N_{betul}}{N_{betul} + N_{lebihan-jana}}$$

LEBIHAN-JANA

Lebihan-jana mengukur berapa banyak lebihan maklumat atau elemen yang ditemui di dalam output OOP Tool yang mana ia tidak ada di dalam *jawapan sebenar*. Formula yang digunakan bagi mengukur lebihan-jana adalah seperti berikut:

$$Lebihan-jana = \frac{N_{lebihan-jana}}{N_{jawapan\_sebenar}}$$

Berdasarkan formula di atas, $N_{lebihan\text{-}jana}$ mewakilkan bilangan lebihan-jana yang dihasilkan oleh OOP Tool dan $N_{jawapan\_sebenar}$ pula mewakilkan bilangan yang dihasilkan oleh penganalisis atau *jawapan sebenar*.

KURANGAN-JANA

Kurangan-jana mengukur berapa banyak maklumat atau elemem yang tidak dikenalpasti di mana elemen tersebut wujud di dalam *jawapan sebenar* tetapi tidak ada di dalam output OOP Tool. Formula yang digunakan adalah seperti yang ditunjukkan di bawah:

$$Kurangan\ jana = \frac{N_{kurangan-jana}}{N_{jawapan\_sebenar}}$$

di mana $N_{kurangan\text{-}jana}$ mewakilkan bilangan elemen yang hilang dan $N_{jawapan\_sebenar}$ pula mewakilkan bilangan yang dihasilkan oleh penganalisis atau *jawapan sebenar*.

KEPUTUSAN PENILAIAN

Keputusan penilaian yang diperolehi adalah berdasarkan kepada pengujian yang dibuat ke atas OOP Tool. Sebanyak 25 soalan pengaturcaraan berorientasikan objek telah digunakan bagi melakukan proses pengujian tersebut. Jadual 3 berikut menunjukkan keputusan penilaian yang telah diperolehi.

Berdasarkan keputusan yang dipaparkan dalam Jadual 3, didapati bahawa OOP Tool menghasilkan nilai dapatan semula yang tinggi iaitu 90.5%. OOP Tool berjaya menghasilkan elemen asas pengaturcaraan berorientasikan objek iaitu kelas, pembolehubah dan fungsi/metod yang releven iaitu 100% tepat dengan *jawapan sebenar* dalam 14 set data daripada 25 set data keseluruhan. Nilai yang diperoleh bagi kejituan pula adalah sebanyak 76.3%, Kurangan-jana sebanyak 9.5% dan Lebihan-jana sebanyak 37.7%. Merujuk kepada jadual tersebut juga, keputusan bagi kejituan yang paling tinggi adalah 100% di mana sebanyak 7 set data telah memperoleh nilai kejituan tertinggi tersebut. Ini menunjukkan bahawa semua elemen asas pengaturcaraan berorientasikan objek yang dihasilkan oleh OOP Tool adalah betul. Walau bagaimanapun, terdapat satu set data iaitu Test4 telah menghasilkan nilai kejituan yang paling rendah iaitu sebanyak 33.3%.

JADUAL 3. Keputusan penilaian

Set Data	Dapatan semula (%)	Kejituan (%)	Kurangan-jana (%)	Lebihan-jana (%)
Test 1	75.0	75.0	25.0	25.0
Test 2	100.0	63.6	0.0	57.1
Test 3	77.8	70.0	22.2	33.3
Test 4	100.0	33.3	0.0	200.0
Test 5	100.0	75.0	0.0	33.3
Test 6	100.0	50.0	0.0	100.0
Test 7	100.0	100.0	0.0	0.0
Test 8	75.0	60.0	25.0	50.0
Test 9	66.7	40.0	33.3	100.0
Test 10	100.0	100.0	0.0	0.0
Test 11	100.0	75.0	0.0	33.3
Test 12	100.0	60.0	0.0	66.7
Test 13	100.0	75.0	0.0	33.3
Test 14	83.3	71.4	16.7	33.3
Test 15	88.9	72.7	11.1	33.3
Test 16	80.0	100.0	20.0	0.0
Test 17	100.0	75.0	0.0	33.3
Test 18	100.0	80.0	0.0	25.0
Test 19	100.0	100.0	0.0	0.0
Test 20	85.7	100.0	14.3	0.0
Test 21	100.0	88.9	0.0	12.5
Test 22	66.7	100.0	33.3	0.0
Test 23	80.0	100.0	20.0	0.0
Test 24	100.0	71.4	0.0	40.0
Test 25	83.3	71.4	16.7	33.3
PURATA	90.5	76.3	9.5	37.7

Ini adalah kerana terdapat 4 elemen telah Lebihan-jana; melebihi bilangan elemen yang betul iaitu sebanyak 2. Set data Test 4 ini juga telah menyumbangkan nilai Lebihan-jana yang paling tinggi iaitu 200% kerana bilangan yang diperolehnya adalah 2 kali ganda daripada bilangan betul.

PERBINCANGAN

Keputusan penilaian yang diperoleh daripada proses pengujian OOP Tool dibandingkan dengan hasil keputusan daripada kajian terdahulu yang berkaitan atau setara dengan kajian yang dilakukan ini. Kajian terdahulu yang dibuat perbandingan ialah CM-Builder. Ini kerana kajian yang dilakukan dalam CM-Builder juga adalah berkaitan dengan analisis berorientasikan objek. Jadual 4 menunjukkan rumusan keputusan penilaian bagi sistem CM-Builder dan OOP Tool. Keputusan keseluruhan yang dihasilkan oleh CM-Builder ialah 73% bagi *recall*, 66% bagi *precision* dan 62% bagi *overspesification* atau lebihan-jana (*over-generated*).

Menggunakan pengukuran yang sama, keputusan yang dihasilkan oleh OOP Tool adalah 90.5% bagi dapatan semula (*recall*), 76.3% bagi kejituan (*precision*) dan 37.7% bagi lebihan-jana (*over-generated*). Jika dibandingkan dengan CM-Builder, keputusan yang dihasilkan oleh OOP Tool adalah lebih baik kerana nilai betul atau tepat yang dijana oleh sistem adalah lebih tinggi. Selain daripada itu, nilai lebihan-jana yang dihasilkan juga adalah lebih rendah. Keputusan penilaian yang diperoleh menggunakan OOP Tool adalah berdasarkan soalan pengaturcaraan berorientasikan objek peringkat permulaan bagi pelajar tahun satu di pusat pengajian tinggi.

JADUAL 4. Rumusan penilaian antara CM-Builder dengan OOP Tool

Sistem	Keputusan Penilaian			
	Recall	Precision	Lebihan-jana	Kurangan-jana
CM-Builder	73%	66%	62%	–
OOP Tool	90.5%	76.3%	37.7%	9.5%

KESIMPULAN DAN KAJIAN LANJUTAN

OOP Tool merupakan sebuah prototaip berasaskan heuristik yang dapat menentukan elemen asas bagi proses penulisan pengaturcaraan berorientasikan objek melalui spesifikasi bahasa tabii. Sumbangan ini boleh dilaksanakan dalam pelbagai bidang seperti *Intelligent Tutoring System* (ITS), dapat digunakan dalam proses pengajaran dan pembelajaran bagi mata pelajaran pengaturcaraan berorientasikan objek dan mana-mana aplikasi pemprosesan bahasa tabii untuk analisis berorientasikan objek.

OOP Tool merupakan sebuah prototaip yang dapat menjana elemen asas pengaturcaraan berorientasikan objek setelah soalan pengaturcaraan yang telah dihuraikan (*parse*) dimasukkan ke persekitaran prototaip. Walau bagaimanapun, OOP Tool ini hanya terhad kepada penjanaan tiga elemen asas sahaja iaitu kelas, pembolehubah/atribut dan fungsi/metod. Oleh itu, dicadangkan agar OOP Tool dipertingkatkan lagi kepenggunaannya dari segi penjanaan lebih banyak lagi elemen asas pengaturcaraan berorientasikan objek seperti perwarisan dan polimorfisma.

Selain daripada itu, OOP Tool hanya boleh menerima kemasukan soalan pengaturcaraan berorientasikan objek yang telah dihuraikan dengan menggunakan penghurai *Memory-Based Shallow Parser* (MBSP). Jadi, kajian yang lebih lanjut boleh dilakukan di mana OOP Tool diintegrasikan dengan penghurai (*parser*). Dengan itu, soalan pengaturcaraan boleh dimasukkan secara langsung ke dalam persekitaran OOP Tool.

OOP Tool hanya dapat menerima kemasukan soalan pengaturcaraan berorientasikan objek yang ditulis di dalam Bahasa Inggeris sahaja. Ini adalah kerana MBSP yang dapat menghurai perkataan yang ditulis di dalam Bahasa Inggeris sahaja. Jadi, adalah lebih baik jika pada masa yang akan datang akan ada kajian lanjutan yang dapat mempertingkatkan lagi mutu kepenggunaan OOP Tool di mana soalan pengaturcaraan yang ditulis di dalam Bahasa Melayu juga boleh dimasukkan ke dalam persekitaran OOP Tool dengan menggunakan penghurai Bahasa Melayu bagi menghurai soalan pengaturcaraan tersebut.

OOP Tool ini hanya menyenaraikan nama kelas, pembolehubah/atribut dan fungsi/metod yang boleh digunakan semasa proses penulisan pengaturcaraan. Peralatan (*tool*) ini tidak menspesifikan atau mengkhususkan pembolehubah dan fungsi bagi setiap kelas yang disenaraikan atau ditentukan. Ini bermakna, kajian lanjutan perlu dilakukan di mana OOP Tool boleh menentukan pembolehubah dan fungsi yang khusus bagi setiap kelas yang dikenalpasti.

RUJUKAN

Abbot, A. R. 1983. *Program Design by Informal English Descriptions. Communication of the ACM* 26(11): 882-894.

Booch, G. 1994. *Object oriented analysis and design*. Ed. Ke-2 California: Benjamin/Cummings.

Gonzales, H. G. P. & Kalita, J. K. 2002. *GOOAL: A Graphic Oriented Oriented Analysis Laboratoryi*. OOPsla'02, November 4-8, 2002, Seattle, Washington, USA. ACM 1-5813-626-9/02/0011.

Grishman, R. & Sundheim, B. 1996. *Message Understanding Conference-6: A Brief History*. In Proceedings of the 16th International Conference on Computational Linguisitics, Copenhagen, 466-471.

Harmain, H. M. & Gaizauskas, R. 2003. CM-Builder: A Natural Language-Based CASE Tool for Object-Oriented Analysis. *Automated Software Engineering* 10: 157-181.

Kamus Dewan. 2005. Ed. 4. Kuala Lumpur: Dewan Bahasa dan Pustaka.

Kölling, M. 1999. The problem of teaching object-oriented programming. Part I: Languages. *Journal of Object-Oriented Programming* 11(8): 8-15.

Liu, D., Subramaniam, K., Eberlein, A. & Far, B. H. 2004. *Natural Language Requirements Analysis and Class Model Generation Using UCDA*. IEA/AIE 2004, LNAI 3029: 295-304.

Mala, G. S. A. & Uma, G. V. 2006. *Automatic Construction of Object Oriented Design Models (UML Diagrams) from Natural Language Requirements Specification*. PRICAI 2006, LNAI 4099: 1155-1159.

Nanduri, G. & Rugaber, S. 1995. *Requirements Validation via Automated Natural Language Parsing*. Proceedings of the 28th Annual Hawaii International Conference on System Sciences. IEEE 1060-3425/95.

Omar, N, Hanna, P. & Kevitt, P. M. 2004. *Heuristics-based entity-relationship modeling through natural language processing*. Proceeding of Yhe Fifteeth Irish Conference on Artificial Intelligence and Cognitive Science Department.

Overmyer, S. P., Lavoie, B. & Rambow, O. 2001. *Conceptual modeling through Linguistic Analysis Using LIDA*. IEEE 0-7695-1050-7/01.

Normariani A. Razik
Kolej Teknologi Bestari
Putera Jaya, Permaisuri
22100 Setiu, Tetengganu
Noma_ar@yahoo.com

Nazlia Omar
Pusat Pengajian Teknologi Maklumat
Fakulti Teknologi dan Sains Maklumat
Universiti Kebangsaan Malaysia
43600 UKM Bangi, Selangor
no@ftsm.ukm.my