PERKULIAHAN KE 4

Tujuan Instruksional Khusus (TIK)
Mahasiswa mampu :

- Menjelaskan model kognitif yang merepresentasikan user dari sistem interaktif
- Menjelaskan model hirarki yang merepresentasikan tugas user dan struktur goal/tujuan
- Menjelaskan model linguistik yang merepresentasikan grammar sistem user
- Menjelaskan model fisik dan device yang merepresentasikan kemampuan motorik manusia
- Menjelaskan arsitektur kognitif sebagai dasar model kognitif

Pokok Bahasan :

- Model Kognitif
- Model Hirarki
- Model Fisik & Device
- Arsitektur Kognitif

Deskripsi Singkat : bahasan ini tentang model-model user dalam suatu sistem interaksi yang direpresentasikan dalam model kognitif dan terdiri dari model hirarki yang merepresentasikan tugas user dan struktur goal/tujuan, model linguistik yang merepresentasikan grammar sistem user, serta model fisik dan device yang merepresentasikan kemampuan motorik manusia.

Bahan Bacaan : Dix, Alan et.al, ***HUMAN-COMPUTER INTERACTION***, Prentice Hall, Europe, 1993, hal 9-114

Johnson, P., ***HUMAN-COMPUTER INTERACTION : Psychology, Task Analysis and Software Engineering***, McGraw-Hill, England UK, 1992

MODEL USER DALAM DESAIN

Tinjauan

- Model kognitif merepresentasikan user dari sistem interaktif
- Model hirarki merepresentasikan tugas dan struktur tujuan user
- Model linguistik merepresentasikan grammar sistem user
- Model fisik dan device yang merepresentasikan kemampuan motorik manusia
- Arsitektur kognitif adalah dasar semua model kognitif

Pendahuluan

Dalam berbagai disiplin ilmu rekayasa (*engineering*), desainer melakukan pemilihan model untuk digunakan dalam proses desain. Model dapat bersifat:

- Evaluatif jika model tersebut digunakan untuk mengevaluasi desain yang ada.
- Generatif jika model mempunyai kontribusi pada proses desain. Dan dalam praktek, model yang sering digunakan adalah yang bersifat generatif.

Model Kognitif

Model-model yang akan dibahas berikut ini merepresentasikan user pada saat mereka berinteraksi dengan interface sistem. Model ini menyertakan aspek pemahaman, pengetahuan, perhatian ataupun pemrosesan yang dilakukan user. Presentasi model kognitif dibagi dalam kategori berikut :

- Representasi hirarki tugas (*task*) user dan struktur goal : terkait dengan isu formulasi tujuan dan tugas.
- Model linguistik dan gramatik : terkait dengan grammar dari translasi artikulasi dan bagaimana pemahamannya oleh user.
- Model tingkat device dan fisik : terkait dengan artikulasi tingkat motorik manusia dan bukan tingkat pemahaman manusia.

Hirarki Tugas dan Goal

Banyak model menggunakan model pemrosesan mental dengan user mencapai tujuan dengan memecahkan sub-tujuan secara divide-and-conquer. Pada bagian ini hanya akan dibahas dua model, yaitu GOMS (*Goals, Operators, Methods and Selections*), dan CCT (*Cognitive Complexity Theory*). Berikut ini dicontohkan tentang membuat laporan penjualan Buku IMK. Untuk mencapai tujuan, kita membaginya menjadi beberapa sub-tujuan.

```
Procedure report
        gather data
                find book names
                        do keywords search of name database
                                <<futher sub-goals>>
                        sift through names & abstracts by hand
                                <<futher sub-goals>>
                search sales database
                        <<futher sub-goals>>
        layout tables and histograms
                <<futher sub-goals>>
        write description
                <<futher sub-goals>>
```

Terdapat beberapa isu penting jika dilakukan analisis penggunaan komputer seperti :

- Dimana kita berhenti. Kita dapat berhenti mendekomposisi tugas jika menyangkut elemen pergerakan motorik manusia atau pada level yang lebih abstrak.
- Dimana kita memulai. Kita dapat memulai analisis di poin yang berbeda pada hirarki goal tertentu.
- Apa yang harus dilakukan ketika ada beberapa solusi pemecahan masalah atau jika solusi terhadap dua masalah saling berhubungan.
- Apa yang harus dilakukan terhadap kesalahan yang terjadi.

GOMS (Goals, Operators, Methods, and Selection)

Model GOMS yang diperkenalkan oleh Card, Moran dan Newell merupakan singkatan dari *Goals, Operators, Methods, and Selection*. Deskripsi model GOMS terdiri dari empat elemen, yaitu :

- ***Goal***, adalah tujuan yang ingin dicapai oleh user
- ***Operator***, merupakan level terendah analisis terdiri atas tindakan dasar yang harus dilakukan user dalam menggunakan sistem.
- ***Methods.*** Seperti yang telah dijelaskan bahwa ada beberapa cara untuk membagi tujuan kedalam beberapa sub-tujuan. Dekomposisi tujuan tersebut disebut sebagai methods.

  Contoh:

  GOAL : ICONIZE-WINDOW
  . [select GOAL : USE-CLOSE-METHOD
  . . MOVE-MOUSE-TO-WINDOW-HEADER
  . . POP-UP-MENU
  . . CLICK-OVER-CLOSE-OPTION
  . GOAL : USE-L7-METHOD
  . . PRESS-L7-KEY]

  Tanda titik menunjukkan hirarki level tujuan / goal.

- ***Selection***, merupakan pilihan terhadap metode yang ada. GOMS tidak membiarkan pilihan menjadi random, namun lebih dapat diprediksikan. Namun secara umum, hal tersebut bergantung pada user, kondisi sistem dan detail tujuan.

  Sebagai contoh, user Sam tidak pernah menggunakan L7-METHOD kecuali jika dia memainkan sebuah game yang bernama ‘blocks’. Maka GOMS memasukkan dalam aturan selection :

  User Sam :

  Rule 1 : Use the CLOSE-METHOD unless another rule applies.
  Rule 2 : If the application is ‘blocks’ use the L7-METHOD

Analisis GOMS umumnya terdiri dari satu high-level goal, kemudian didekomposisi menjadi deretan unit tugas (*task*), dan selanjutnya dapat didekomposisi lagi sampai pada level operator dasar. Dekomposisi tujuan antara tugas keseluruhan dan unit tugas melibatkan pemahaman mengenai strategi pemecahan masalah oleh user dan domain aplikasi secara detail. Bentuk deskripsi high-level-goal ini nantinya diadopsi selama proses analisis tugas.

Analisis struktur tujuan GOMS dapat digunakan untuk mengukur kinerja. Kedalaman tumpukan struktur tujuan dapat digunakan untuk mengestimasi kebutuhan memori jangka-pendek. Selection dapat diuji untuk keakuratannya terhadap jejak user (*user's traces*) dan perubahan respon. GOMS merupakan metode yang baik untuk mendeskripsikan bagaimana seorang ahli melakukan pekerjaannya dan jika digabung dengan moel fisik dan device dapat digunakan untuk memprediksikan kinerja user dari aspek waktu eksekusi.

Cognitive Complexity Theory (CCT)

Cognitive Complexity Theory (CCT) yang diperkenalkan oleh Kieras dan Polson ini dimulai dengan premis dasar dekomposisi tujuan dari GOMS dan disempurnakan modelnya agar lebih dapat diprediksi. CCT memiliki dua deskripsi paralel, satu adalah tujuan user dan yang lainnya adalah sistem komputer atau disebut device pada CCT. Deskripsi tujuan user berdasarkan hirarki tujuan yang mirip dengan GOMS, tetapi diekspresikan menggunakan *production rules*. Production rules merupakan sekumpulan aturan dengan bentuk :

If *kondisi* then *aksi*

Dengan kondisi adalah pernyataan tentang isi memori kerja. Jika kondisi bernilai benar maka production rule dijalankan. Sedangkan aksi dapat terdiri dari satu atau lebih aksi elementer yang mungkin mengubah memori kerja atau berupa aksi eksternal seperti keystroke. Sebagai contoh adalah tugas editing menggunakan editor 'vi' UNIX untuk mengoreksi spasi antar kata.

Production rules CCT:

```
(SELECT-INSERT-SPACE
IF (AND (TEST-GOAL perform unit task)
        ( TEST-TEXT task is insert space)
        (NOT (TEST-GOAL insert space))
        (NOT (TEST-NOTE executing insert space)))
THEN (  (ADD-GOAL insert space)
        (ADD-NOTE executing insert space)
        (LOOK-TEXT task is at %LINE %COL)))

(INSERT-SPACE-DONE
IF (AND (TEST-GOAL perform unit task)
        (TEST-NOTE executing insert space)
        (NOT (TEST-GOAL insert space)))
THEN (  (DELETE-NOTE executing insert space)
        (DELETE-GOAL perform unit task)
        (UNBIND %LINE %COL))

(INSERT-SPACE-1
IF (AND (TEST-GOAL insert space)
        (NOT (TEST-GOAL move cursor))
        (NOT (TEST-CURSOR %LINE %COL)))
THEN (  (ADD-GOAL move cursor to %LINE %COL)))

(INSERT-SPACE-2
IF (AND (TEST-GOAL insert space)
        (TEST-CURSOR %LINE %COL))
THEN (  (DO-KEYSTROKE 'I')
        (DO-KEYSTROKE  SPACE)
        (DO-KEYSTROKE  ESC)
        (DELETE-GOAL insert space)))
```

Untuk mengetahui cara kerja rules, anggap user baru saja melihat karakter yang salah dan isi dari memori kerja adalah :

(GOAL perform unit task)
(TEXT task is insert space)
(TEXT task is at 5 23)
(CURSOR 8 7)

Melihat pada empat rule yang ada (SELECT-INSERT-SPACE, INSERT-SPACE-DONE, INSERT-SPACE-1, INSERT-SPACE-2) maka hanya SELECT-INSERT-SPACE yang dipenuhi, sehingga isi memori kerja setelah rule tersebut dijalankan adalah :

(GOAL perform unit task)
(TEXT task is insert space)
(TEXT task is at 5 23)
(NOTE executing insert space)
(GOAL insert space)
(LINE 5)
(COL 23)
(CURSOR 8 7)

Kemudian dijalankan berturut-turut rule INSERT-SPACE-1, INSERT-SPACE-2, dan INSERT-SPACE-DONE hingga tugas mengoreksi spasi dapat diselesaikan.

Rule dalam CCT tidak selalu merepresentasikan kinerja yang bebas dari kesalahan (*error*). Aturan tersebut dapat digunakan untuk menerangkan fenomena error meskipun tidak dapat memprediksikannya. Sebagai contoh rule INSERT-SPACE-2 untuk menyisipkan spasi tidak mengecek modus editor yang sedang digunakan.

Rule CCT dapat menggambarkan rencana (*plan*) yang kompleks dibandingkan dengan hirarki sekuensial pada GOMS. Aktifitas yang kontinyu dari semua *production rule* memungkinkan untuk merepresentasikan rencana yang berkesinambungan. Sebagai contoh, pada CCT dapat dibuat satu set *production rule* untuk menulis sebuah buku dan satu set lainnya untuk membuat teh. Kedua jenis *production rule* ini dapat aktif secara simultan sehingga memungkinkan si penulis menulis buku sambil minum teh.

Secara umum, semakin banyak *production rules* dalam CCT semakin sulit suatu interface untuk dipelajari. Selain itu terdapat beberapa masalah pada CCT, yaitu :

- Semakin detail deskripsinya, size deskripsi dari satu bagian interface dapat menjadi sangat besar. Lebih jauh, dimungkinkan terdapat beberapa cara untuk merepresentasikan perilakuk user dan interace yang sama sehingga mengakibatkan perbedaan hasil pengukuran.
- Pemilihan notasi yang digunakan. Muncul pertanyaan kapan notasi tertentu yang dipilih menjadi suatu hal yang penting / kritis. Sesorang dapat saja memilih untuk merepresentasikan sistem dengan notasi yang berbeda. Notasi yang berbeda dapat mengakibatkan perbedaan pengukuran.

  Contoh: pada deskripsi sebelumnya (NOTE executing insert space) hanya digunakan untuk membuat rule INSERT-SPACE-DONE dijalankan pada waktu yang tepat. Disini tidak jelas sama sekali signifikansi kognitifnya.
- CCT adalah alat perekayasaan (*engineering tool*) dengan pengukuran kemudahan dipelajari (*learnability*) dan tingkat kesulitan (*difficulty*) secara garis besar digabung dengan dekripsi detail dari perilaku (*behaviour*) user.

Model Linguistik

Interaksi user dengan komputer dapat dipandang dari sisi bahasa (*language*), sehingga wajar saja jika beberapa formalisasi model menggunakan konsep ini. Grammar BNF paling sering digunakan untuk menspesifikasikan dialog. Model yang mirip dengan notasi dialog ini digunakan untuk memahami perilaku user dan menganalisis kesulitan kognitif dari interface.

Backus-Naur Form (BNF)

Salah satu representasi dari pendekatan linguistik adalah Backus-Naur Form yang diperkenalkan oleh Reisner untuk mendeskripsikan tata bahasa (grammar) dari dialog. Model ini memandang dialog hanya pada level sintaksis dan mengabaikan semantik dari bahasa tersebut. BNF banyak digunakan untuk menspesifikasikan sintaks dari bahasa pemrograman komputer dan banyak

sistem dialog. Sebagai contoh sebuah sistem grafik yang memiliki fungsi menggambar garis.

```
draw-line          ::=   select-line + choose-points
                                    + last-point
select-line        ::=   position-mouse + CLICK-MOUSE
choose-points      ::=   choose-one
                         | choose-one + choose-points
choose-one         ::=   position-mouse + CLICK-MOUSE
last-point         ::=   position-mouse + DOUBLE-CLICK-MOUSE
position-mouse::=        empty | MOVE-MOUSE + position-mouse
```

Nama yang digunakan pada deskripsi terdiri dari dua jenis, yaitu ***non-terminal*** yang dituliskan dengan huruf kecil, dan ***terminal*** yang dituliskan dengan huruf kapital. ***Terminal*** merepresentasikan level terrendah dari perilaku user seperti menekan tombol, memindahkan mouse, atau meng-klik tombol mouse, dan sebagainya. Sedangkan ***non-terminal*** adalah abstraksi level tinggi yang dapat terdiri dari non-terminal lainnya dan terminal dengan format :

```
name ::=    expression
```

Simbol **: :=** berarti didefinisikan sebagai. Hanya non-terminal yang boleh berada di bagian kiri dari simbol tersebut. Bagian kanan dibentuk dengan menggunakan operator **'+'** yang berarti berurutan (*sequence*) dan **'|'** yang berarti pilihan (*choice*).

Deskripsi BNF dari suatu interface dapat dianalisis dengan berbagai cara, salah satunya adalah dengan mengukur jumlah rules dan operatornya. Semakin banyak jumlah rule-nya, semakin rumit interface tersebut. Dan hal ini bergantung pada cara pendeskripsian interface, karena bisa saja aturan untuk perilaku user yang disama dideskripsikan dengan cara yang berbeda.

Selain dari pengukuran kompleksitas bahasa secara keseluruhan, BNF dapat digunakan untuk menentukan berapa banyak tindakan dasar yang dibutuhkan dalam tugas tertentu, dan didapatkan estimasi kasar dari kesulitan

tugas tersebut. Deskripsi BNF hanya digunakan untuk merepresentasikan aksi yang dilakukan user bukan persepsi user terhadap sistem.

Task-Action Grammar (TAG)

BNF mengabaikan konsistensi struktur bahasa dan penggunaan nama perintah serta huruf. Task-Action Grammar (TAG) mencoba mengatasi kekurangan tersebut dengan menyertakan grammar berparameter untuk menekankan konsistensi serta menyimpan knowledge user. Contoh berikut ini adalah untuk mengilustrasikan konsistensi menggunakan tiga buah perintah UNIX, yaitu cp (menyalin file), mv (memindahkan file), dan ln (link file) yang masing-masing memiliki dua bentuk.

```
copy      ::=  'cp' + filename + filename
             | 'cp' + filename + directory
move      ::=  'mv' + filename + filename
             | 'mv' + filename + directory
link      ::=  'ln' + filename + filename
             | 'ln' + filename + directory
```

BNF tidak dapat membedakan konsistensi perintah dan inkonsistensi alternatif, misalkan jika 'ln' mengambil argumen direktori lebih dahulu. TAG didesain untuk menyatakan konsistensi semacam ini, dan deskripsinya diubah menjadi :

```
file-op [Op]                 :=  command-op[Op]+ filename + filename
                               | command-op[Op] + filename + directory
command-op[Op=copy]  := 'cp'
command-op[Op=move]  := 'mv'
command-op[Op=link]  := 'ln'
```

Bentuk tersebut memperlihatkan konsistensi perintah dan mencerminkan deskripsi tekstual aslinya. Pengukuran kompleksitas bahasa menggunakan deskripsi TAG akan lebih baik dalam hal memprediksikan *learning* dan *performance* dibandingakan BNF.

Selain konsistensi, TAG juga ditujukan untuk menyimpan pengetahuan (*knowledge*). Sebagai contoh kita memiliki dua buah command line interface untuk menggerakan kura-kura mekanik di atas lantai.

Command interface 1

movement [Direction]
:= command[Direction] + distance + RETURN
command[Direction=forward] := 'go 395'
command[Direction=backward] := 'go 013'
command[Direction=left] := 'go 712'
command[Direction=right] := 'go 956'

Command interface 2

movement [Direction]
:= command[Direction] + distance + RETURN
command[Direction=forward] := 'FORWARD'
command[Direction=backward] := 'BACKWARD'
command[Direction=left] := 'LEFT'
command[Direction=right] := 'RIGHT'

Jelas terlihat bahwa interface kedua lebih disukai. TAG menambahkan bentuk khusus *known-item* yang digunakan untuk menginformasi-kan ke user bahwa inputnya sudah diketahui secara umum dan tidak perlu dipelajari jika user akan menggunakan sistem. Dengan menggunakan bentuk ini, Interface kedua dapat ditulis ulang sebagai berikut:

Command interface 2

movement [Direction]
:= command[Direction] + distance + RETURN
command[Direction] := known-item[Type = word, Direction]
command[Direction=forward] := 'FORWARD'
command[Direction=backward] := 'BACKWARD'
command[Direction=left] := 'LEFT'
command[Direction=right] := 'RIGHT'

Model Fisik dan Device

Keystroke Level Model (KLM)

Dibandingkan dengan model-model pemahaman kognitif, sistem motorik manusia lebih mudah untuk dipahami. Keystroke Level Model (KLM) menggunakan sistem motorik tersebut sebagai dasar untuk memprediksikan kinerja user. Model ini merepresentasikan sebuah unit tugas dalam interaksi seperti eksekusi dari urutan perintah sederhana. Tugas yang kompleks akan dibagi menjadi sub-tugas sebelum dipetakan pada aksi fisik. Tugas dapat didekomposisi menjadi dua fase, yaitu :

- Akuisisi tugas, ketika user membangun representasi mental dari tugas
- Eksekusi tugas menggunakan fasilitas sistem

KLM hanya memberikan prediksi untuk kegiatan pada tahap berikutnya. Pada tahap akuisisi, user memutuskan bagaimana cara melakukan tugas menggunakan sistem. KLM berkaitan dengan model GOMS dan dapat dikategorikan sebagai model GOMS tingkat terendah.

Model KLM mendekomposisi fase eksekusi menjadi lima operator motorik fisik, operator mental dan operator respon sistem berikut ini :

K keystroking
B menekan tombol mouse
P pointing, menggerakkan mouse (atau sejenis) ke target
H Homing, perpindahan tangan antar mouse dan keyboard
D menggambar garis dengan mouse
M persiapan mental untuk tindakan fisik
R respon sistem, dapat diabaikan jika user tidak perlu menunggu untuk itu

Eksekusi sebuah tugas akan melibatkan berbagai operator tersebut. Sebagai contoh tugas mengedit karakter tunggal yang salah didekomposisi menjadi :

1. memindahkan tangan ke mouse	H[mouse]
2. Meletakkan cursor setelah karakter yang salah	PB[LEFT]
3. Kembali ke keyboard	H[keyboard]
4. Hapus kerakter	MK[DELETE]

5. Ketik koreksi K[char]
6. Mereposisi ke insertion point H[mouse]MPB[LEFT]

Model ini dapat memprediksikan waktu yang diperlukan pada fase eksekusi dengan menjumlahkan waktu masing-masing komponen aktifitas. Maka waktu yang dibutuhkan untuk tugas di atas adalah :

$$T_{execute} = T_K + T_B + T_P + T_H + T_D + T_M + T_R$$
$$= 2t_K + 2t_B + 2t_P + 3t_H + 0 + 2t_M + 0$$

Berikut ini adalah contoh waktu eksekusi untuk beberapa operator KLM diadaptasi dari Card, Moran, dan Newell, 1983.

Tabel 4.1 Waktu Eksekusi Untuk Beberapa Operator KLM Diadaptasi Dari Card, Moran, Dan Newell

Operator	Remarks	Time (sec)
K	Press Key	
	good typist (90 wpm)	0.12
	poor typist (40 wpm)	0.28
	non-typist	1.20
B	Mouse button press	
	down or up	0.10
	click	0.20
P	Point with mouse	
	Fitts' Law	$0.1\log_2(D/S + 0.5)$
	average movement	1.10
H	Home hands to and from keyboard	0.40
D	Drawing – domain-dependent	-
M	Mentally prepare	1.35
R	Response from system - measure	-

D = jarak target S = ukuran target

Dengan menggunakan tabel di atas, waktu eksekusi tugas mengedit karakter tunggal yang salah menjadi :

$$T_{execute} = T_K + T_B + T_P + T_H + T_D + T_M + T_R$$
$$= 2t_K + 2t_B + 2t_P + 3t_H + 0 + 2t_M + 0$$
$$= 2 * 0.28 + 2 * 0.20 + 2 * 1.10 + 3 * 0.40 + 2 * 1.35$$
$$= 7.06 \text{ second}$$

Jika dua metode pada model GOMS dideskripsikan operatornya maka akan menjadi :

L7_METHOD **H**[to keyboard] **M K**[L7 function key]
CLOSE_METHOD **P**[to menu bar] **B**[LEFT down] **M P**[to option] **B**[LEFT up]

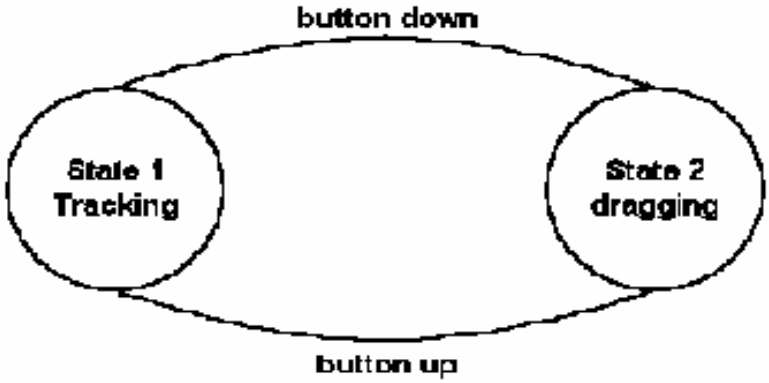
L7_METHOD = 0.4 + 1.35 + 0.28
= 2.03 second

CLOSE_METHOD = 1.1 + 0.1 + 1.35 + 1.1 + 0.1
= 3.75 second

Three-State Model

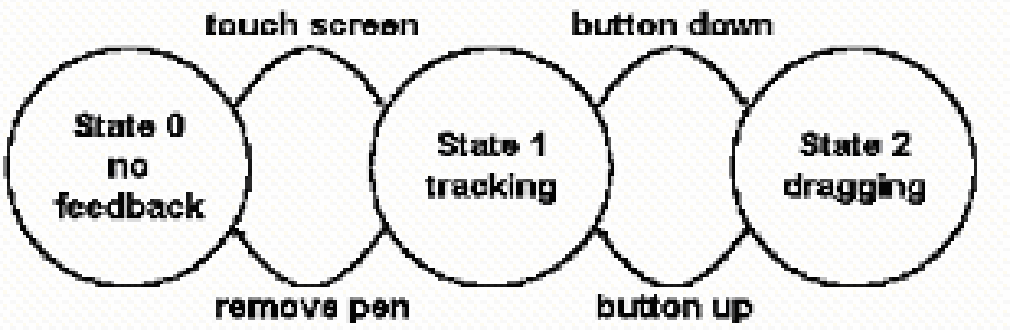
Pada bahasan perkuliahan ke-2 telah dijelaskan bahwa ada berbagai macam device penunjuk yang digunakan selain mouse. Device biasanya dapat dinyatakan equivalen secara logika ditinjau dari level aplikasi, namun berbeda dari karakteristik motor-sensor fisiknya. Oleh karena itu three-state model dibuat untuk mewakili device tersebut.

Sebagai contoh pada device mouse yang perilakunya digambarkan pada gambar 4.1 di bawah ini. Jika menggerakkan mouse tanpa menekan buttonnya maka hanya akan menggerakkan kursor saja. Perilaku ini dinyatakan sebagai tracking dengan nomor state 1. Dengan menekan button mouse dan menggerakkan mouse akan menyebabkan obyek tergeser (*dragged*), dan perilaku ini dinyatakan sebagai state 2.



Gambar 4.1 Transisi Mouse : State 1 dan 2

Demikian juga dengan perilaku light pen yang memiliki button yang digambarkan pada gambar 4.2. Jika buttonnya tidak ditekan maka perilakunya akan seperti mouse pada state 1 (*tracking*), dan jika button ditekan akan berada pada state 2 (*dragging*). Light pen memiliki state ketiga yaitu jika light pen tidak menyentuh layar dan didefinisikan sebagai state 0.



Gambar 4.2 Transisi Light Pen : Three State

Model three state ini mengkarakteristikan state dari sebuah device berdasarkan jumlah input device tersebut bagi sistem. Hukum Fitt (Fitt's Law) mendefinisikan konstanta waktu yang berbeda untuk device yang berbeda. Hukum Fitt menyatakan bahwa waktu yang dibutuhkan untuk berpindah ke target yang berukuran S dengan jarak D adalah :

$$a + b \log_2 (D/S + 1)$$

Konstanta *a* dan *b* bergantung pada alat penunjuk (*pointing device*) yang digunakan dan keahlian yang dimiliki oleh user untuk menggunakan device tersebut. Namun model three-state mengindikasikan bahwa konstanta-konstanta tersebut juga ditentukan oleh state dari device. Tabel 4.2 berikut menunjukkan konstanta *a* dan *b* untuk pointing device mouse dan trackball menurut Hukum Fitt.

Tabel 4.2 Konstanta *a* dan *b* Untuk Pointing Device Mouse Dan Trackball Menurut Hukum Fitt

Device	a (ms)	b (ms/bit)
Pointing (state 1)		
Mouse	-107	223
Trackball	75	300
Dragging (state 2)		
Mouse	135	249
Trackball	-349	688

Jika kita kalkulasikan CLOSE_METHOD menggunakan operator KLM maka hasilnya menjadi :

Mouse

P[to menu bar] = -107 + 223log2(11) = 664 ms

P[to option] = 135 + 249log2(5) = 713 ms

Trackerball

P[to menu bar] = 75 + 300log2(11) = 1113 ms

P[to option] = -349 + 688log2(5) = 1248 ms

Arsitektur Kognitif

Bentuk formalisme yang sudah kita bahas sebelumnya, memperlihatkan model implisit dan eksplisit bagaimana user melakukan proses kognitif dalam melaksanakan tugasnya. Seperti GOMS yang menggunakan konsep pembagian tujuan (*goal*) menjadi sub-tujuan (*sub-goal*) untuk mengenali dan memecahkan masalah, CCT membedakan *short-term memory* dan *long-term memory* dengan *production rules* disimpan pada *long-term memory* dan dicocokan dengan isi yang ada di *short-term memory* untuk menentukan aksi mana yang dijalankan, dan sebagainya.

Notasi hirarki dan linguistik cenderung berorientasi pada dialog yang sempurna pada sisi user. Metode-metode tersebut mengukur kompleksitas dialog yang dimaksud namun tidak memperhatikan perbedaan dialog yang ada dengan urutan yang optimal atau seharusnya. Pada model arsitektur yang dibahas di bagian ini, prediksi dan pemahaman terhadap kesalahan merupakan fokus dari analisis yang dilakukan.

Model Problem Space

Rational behaviour (perilaku rasional) didefinisikan sebagai perilaku yang dibentuk untuk mencapai tujuan (*goal*) khusus tertentu. Adanya elemen rasionalitas ini digunakan untuk membedakan antara perilaku sistem cerdas (*intelligent*) dengan mesin (*machine-like*). Dalam bidang *artificial intelligent* (AI), sistem yang memiliki karakter *rational behaviour* dikenal sebagai *knowledge level system*.

Dalam sebuah *knowledge level system* terdapat *agent* yang memiliki pengetahuan mengenai diri dan lingkungannya, termasuk tujuan yang akan dicapai. *Agent* dapat melaksanakan suatu aksi tertentu dan menangkap informasi perubahan lingkungannya. Saat *agent* berperilaku atau melakukan suatu aksi dalam lingkungannya, ia mengubah kondisi lingkungan dan pengetahuan yang dimilikinya. Kita dapat melihat perilaku keseluruhan *knowledge level system* sebagai serangkaian *state* (kondisi) lingkungan dan *agent* yang berubah seiring dengan berjalannya proses. Tujuan *agent* dapat

didefinisikan sebagai pilihan (*preference*) dari semua rangkaian *state agent* atau lingkungan yang mungkin.

Model yang kontras dengan *rational behaviour* adalah model non-rasional yang umum dipakai sebagai model komputasi. Dalam bidang ilmu komputer, masalah / problem dideskripsikan sebagai suatu proses pencarian diantara sekumpulan *state* yang mungkin mulai dari *state* awal ke *state* yang lainnya melalui suatu operasi atau aksi untuk mencapai tujuan.

Model komputasi *problem space* diadaptasikan dari model pemecahan masalah yang dikemukakan oleh Newell dan Simon dari Carnegie Mellon University. *Problem space* terdiri dari sekumpulan *state* dan operasi yang dilakukan terhadap *state*. Perilaku dalam model ini merupakan proses yang terdiri dari dua tahap. Pertama, operator dipilih berdasarkan *state* yang ada kemudian diaplikasikan terhadap *state* untuk menghasilkan *state* baru. *Problem space* harus merepresentasikan *rational behaviour* sehingga model ini harus memiliki karakter tujuan seperti halnya *agent*. *Problem space* merepresentasikan tujuan dengan mendefinisikan *state* yang diinginkan sebagai bagian (subset) dari semua *state* yang mungkin. Setelah *state* awal ditetapkan, maka tugas yang dilakukan adalah menemukan rangkaian operasi yang membentuk *path* / jalur *state* menuju *state* yang diinginkan. Sehingga dapat disimpulkan terdapat empat aktifitas yang dilakukan pada model *problem space*, yaitu formulasi tujuan (*goal formulation*), pemilihan operasi (*operation selection*), aplikasi operasi (*operation application*), dan pencapaian tujuan (*goal completion*).

Siklus aktifitas pada *problem space* dimulai dari indetifikasi perubahan lingkungan eksternal yang relevan terhadap tujuan oleh proses *goal formulation* yang kemudian didefinisikan terhadap kelompok *state* yang diinginkan dan *state* awal. Proses *operation selection* menyarankan operasi mana yang dapat dilakukan terhadap suatu *state* dan menjadikannya lebih dekat ke *state* tujuan. Proses *operation application* menjalankan operasi, mengubah *state* dan keadaan lingkungan. Jika *state* yang dinginkan sudah tercapai, proses *goal completion* menonaktifkan *agent*.

Kelebihan pada arsitektur model ini adalah rekursifnya. Aktifitas pada proses manapun di model ini hanya akan dijalankan jika pengetahuan (*knowledge*) yang dibutuhkan tersedia. Sebagai contoh, untuk menentukan operasi mana yang akan dipilih diperlukan, dibutuhkan pengetahuan mengenai *state* yang terkini dan lingkungannya. Jika informasi tersebut tidak tersedia, maka akan muncul *problem space* lagi dari tugas pencarian informasi tersebut dan seterusnya hingga membentuk struktur rekursif.

Salah satu contoh arsitektur *problem space* adalah SOAR yang membentuk *programmable user models* (PUM). Desainer memberikan deskripsi prosedur tugas yang akan dijalankan kemudian analisis terhadap prosedur tersebut akan menghasilkan pengetahuan yang diperlukan user untuk melaksanakan tugas tersebut. Pengetahuan ini dikodekan pada arsitektur SOAR dan menghasilkan model user (PUM). Dengan PUM, *problem space* untuk mencapai tujuan dapat dianalisis untuk mengukur beban kognitif suatu prosedur. Tambahan lagi PUM tidak dapat mencapai tujuan jika pengetahuan yang dibutuhkan tidak tersedia. Hal ini menandakan ada kesalahan dalam perancangan sistem interaksi. Dengan demikian, kesalahan dapat diprediksi sebelum implementasi sistem interaksi.

Model Interacting Cognitive Sub-systems (ICS)

Arsitektur kognitif *Interacting Cognitive Sub-systems* (ICS) diperkenalkan oleh Barnard. ICS merupakan model persepsi, kognitif dan aksi, yang berbeda dengan model lainnya. Jika model lain menghasilkan deskripsi user dalam lingkup urutan aksi yang dilakukannya, ICS memberikan pandangan terhadap user secara menyeluruh sebagai mesin pengolah informasi. Penekanannya pada bagaimana mudahnya suatu prosedur tertentu jika mereka dibuat otomatis.

Arsitektur ICS terdiri dari sembilan subsistem aktifitas yang terkoordinasi, yaitu lima subsistem peripheral yang berinteraksi secara fisik dengan dunia luar dan empat subsistem pusat (*central sub-system*) yang berhubungan dengan proses mental. Masing-masing subsistem memiliki struktur generik yang sama dan dideskripsikan dalam lingkup input dan output yang diketikan (*typed input*