PELACAKAN KEMBALI KONTROL-KONTROL OBYEK PADA PENAMBAHAN OBYEK SECARA DINAMIS BERBASIS VISUAL BASIC STUDI KASUS ANALISA HIRARKI TUGAS

Dwi Sunaryono

Jurusan Teknik Informatika, Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember (ITS) - Surabaya
Kampus ITS, Jl. Raya ITS, Sukolilo – Surabaya 60111
Tel. + 62 31 5939214, Fax + 62 31 5939363
Email : dwi@its-sby.edu

ABSTRAK

Pembentukan HTA(Hierarchical Task Analysis) pada dunia nyata dapat menentukan urutan tugas yang harus diselesaikan dan beberapa jalan / cara yang dapat dicapai baik secara horizontal maupun vertikal. Pendekatan HTA dapat dipetakan kedalam logika komputer berupa alur tugas dari setiap obyek di komputer yang membentuk hirarki tugas dari masing-masing obyek yang akan dikerjakan oleh komputer. Pekerjaan ini mirip dengan pembuatan strategi dan kemudian hasilnya terlihat secara visual, sehingga memungkinkan terjadinya refresh engineering tanpa harus melakukan kompilasi program. Pekerjaan tersebut terdiri dari : pembuatan aturan untuk masing-masing obyek dan cara penerjemahannya (interpreter).

Setiap obyek yang dibuat diberi urutan tugas yang berkaitan dengan event, property dan method dari suatu obyek yang akan dikerjakan. Perencanaan obyek tersebut tersimpan ke dalam database sehingga untuk pengerjaan akan merefer ke aturan yang ada di database tersebut yang kemudian di load ke memory sama halnya ketika suatu obyek di instanciate.Kontrol-kontrol obyek setelah terbentuk (instance) akan terlepas sehingga diperlukan pelacakan kembali obyek tersebut. Pelacakan ini akan sangat berguna untuk menangkap event yang terjadi. Hasil dari penangakapan event bisa dipakai sebagai method dari obyek tersebut atau event / method ke obyek lain

Kata kunci : HTA,API, *tugas,event,property,method, kontrol,hWnd,hook*

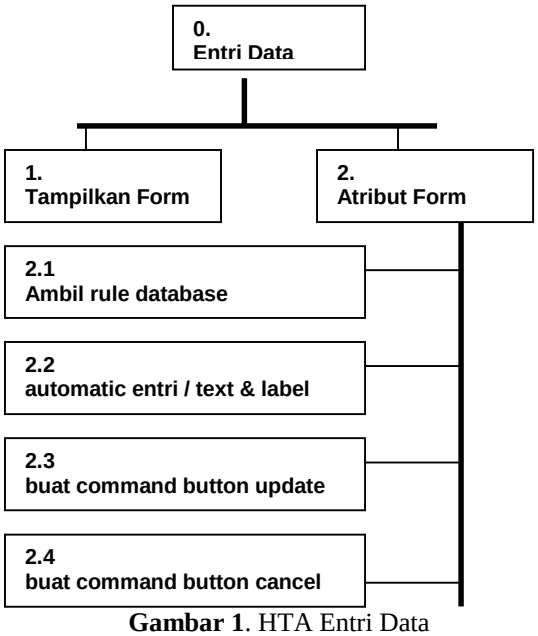
1. PENDAHULUAN

HTA (*Hierarchical Task Analysis)* merupakan salah satu topik pada dunia Interaksi Manusia dan Komputer. Tugas adalah sekumpulan *tugas* yang dikerjakan. Suatu penyelesaian *tugas* terdiri dari dekomposisi tugas dan urutan kelakuannya [1] Hasil dari HTA berupa sekumpulan tugas dan subtugas yang tersusun secara hirarki dan perencanaan/strategi yang melukiskan semua kondisi dari pekerjaan tugasnya[2].

Pelevelan tugas menunjukkan hirarki tugas yang tersusun secara berjenjang. Pada kasus text, maka bisa di tulis sebagai identasi yang menunjukkan level tugas, sedangkan pada aturan obyek, berupa pemetaan kode

Contoh hirarki tugas :

0.Entri Data
  1.Tampilkan Form
  2. Attribut form
    2.1 Ambil rule database
    2.2 Tampilkan automatic entri / text & label
    2.3 Buat command button update
    2.4 Buat command button cancel



Gambar 1. HTA Entri Data

Goal : Entri Data
Rencana : 1,2.1,2.2,2.3,2.4
Pada contoh tersebut dapat dikodekan ke dalam komputer sebagai kode

1: 000 : *create intance form*
2.1: 000-001 : *action SQL command*
2.2: 000-002 : *control Add*
2.3: 000-003 : *control Add*

Level diatas dapat menjaga hubungan *parent, child* dan *container* yang ada, sehingga jika diperlukan referensi ke obyek, maka tersedia lengkap semua urutan obyek

Rencana yang terbentuk diatas bersifat sekuensial, pada banyak kasus aplikasi tidak selalu rencana harus sekuensial. Tipe-tipe rencana terdiri dari :

0. Fixed Sequence
1. Optional Task
2. Waiting for events
3. Cycles
4. Time-Sharing
5. Discretionary
6. Mixtures

Semua tipe rencana tersimpan sebagai aturan dalam penyelesesaian tugas dan disimpan kedalam database.

2. PENGANTAR KONTROL OBYEK

2.1. Kontrol Obyek pada design-time

Biasanya seorang *programmer* akan membuat program melalui form dan kemudian dia menggambar form dengan berbagai macam kontrol yang tersedia. Dalam pustaka Visual Basic, kontrol standar terdapat pada pustaka VB,VBA,VBRUN, sebagai contoh control *command button* terdapat pada pustaka VB atau dikenal sebagai *class member of VB.*

Setelah obyek diletakkan pada form, maka secara otomatis tersedia pula *property, method* dan *event*. *Programmer* dapat melakukan setting pada *property*, misalnya Caption = "&Update", hal ini bisa dilakukan pada pada *design-time* maupun *run-time*. Pada kasus *design-time*, obyek memiliki keterbatasan terhadap perubahan yang sifatnya statis dan akan berkembang sesuai dengan banyaknya kasus yang ditangani. Sebagai contoh pada kasus database transaksional akademik sederhana terdapat sejumlah tabel

Tabel master :

- mahasiswa
- matakuliah
- dosen

Tabel Transaksional :

- jadwalPelajaran
- PesertaKuliah

Terdapat 5 tabel yang secara spesifik melakukan operasi yang berbeda. Pada pendekatan *design-time* minimal diperlukan 5 form dan masing-masing form terdapat kontrol obyek dengan sifat yang mungkin berbeda satu sama lain. Jumlah form, obyek dan kode operasi akan semakin besar seiring bertambahnya kasus yang diselesaikan.

2.2. Kontrol Obyek pada run-time

Pendekatan *design-time* lebih menonjolkan aspek visualisasi obyek, tanpa melihat optimasi program dimana terdapat perulangan yang sama baik pada kode program ataupun obyek yang dibuat. Seperti kasus sederhana di atas, adalah sejumlah form yang sama akan di tambahkan (*add*). Pertambahan obyek yang sama akan membuat hasil kompilasi (file *.exe) menjadi besar. Pada sebagian kasus contoh lengkap penyelesaian kasus dengan pendekatan design time rata-rata hampir 500 Kbytes. Kelemahan ini dapat diatasi dengan pembuatan obyek sebagai instan dari kelas

Kontrol obyek pada run-time dikenal sebagai *instanciate of class*.[3]
Dengan deklarasi :

```
NameObyek new as class
```

Maka suatu obyek telah terbuat dari kelas yang tersedia. Obyek yang terbentuk bisa dipakai pada saat run-time yaitu *property* dan *method*-nya, sedangkan *event* tidak tersedia.

Kontrol obyek tidak terlihat secara visual atau tidak muncul di form, tetapi ada dalam bentuk kode-kode program. Pada level operasi, harus ditangkap *event* yang mungkin dari suatu obyek. Penangkapan ini saat *run-time* sangat menentukan urutan operasi yang dikerjakan berupa : *cycles* atau *optional task.*

Memunculkan *event* pada saat *run-time* dipakai perintah deklarasi :

```
WithEvents NameObyek new as class
```

Perintah ini sangat tergantung (*dependent*) terhadap form-nya, sehingga pengurangan jumlah kontrol yang dibuat dengan mengalihkan pada pembuatan saat *run-time* berupa instan kelas, masih membutuhkan form sebagai tempat operasi *event*-nya. Konsekwensi dari banyaknya form, sama halnya saat pemakaian *wizzard form.* Suatu obyek yang merupakan instan dari kelas masih banyak yang sama pada form yang berbeda walaupun memiliki tugas secara spesifik berbeda.

2.2. Pemakaian template

Untuk mengatasi form ataupun kontrol yang sama untuk operasi yang berbeda dapat didekati dengan memanfaatkan *template form*. Ada dua cara pemanfaatan template :

1. *template form*
2. *template userControl*

Metode template form

Penambahan kontrol secara dinamis dapat dilakukan pada saat *run-time*. Pada form kosong, dapat ditambahkan kontrol-kontrol secara dinamis dalam modul, dan secara otomatis *property* dan *method* dapat dilakukan :

Contoh kontrol dinamis :

```
Licenses.Add "MSComctlLib.ListViewCtrl"
Set ctl = controls.Add("MSComctlLib.
    ListViewCtrl", "newctl", templateForm)
ctl.Move 1, 1, 2500, 3500
```

Metode *template form* dapat secara dinamis membuat kontrol-kontrol yang tersedia, namun bagaimana dengan *event*-nya ?. Hampir sama dengan penambahan kontrol obyek pada saat *run-time*, yaitu dapat ditambahkan pada saat deklarasi :

```
With event ctl as VBControlDynamic
```

Persoalan yang timbul hampir mirip dengan pembuatan kontrol saat *run-time* yaitu sangat bergantung dengan form-nya. Kita tidak bisa membuat kode tambahan didalam form saat *run-time*, sehingga penambahan kontrol dinamis untuk memegang (*hold*) beberapa *event* yang terjadi tidak dapat dilakukan.

Metode UserControl

UserControl adalah suatu upaya untuk membungkus (*encapsulation)* obyek yang telah ada dan tetap membiarkan event yang terjadi pada obyek dapat digabung dengan *userControl*. Metode ini sangat membantu untuk memegang *event* yang terjadi pada suatu obyek. Beberapa *method* dan *property* dan dimasukkan sebagai *ambient property* dan *extender object*. Pengambil alih (*override)* suatu *property* maupun *method* oleh *userControl* seperti *method intrinsic* pada suatu obyek diambil alih oleh *userControl.*

Tiga alasan pemakaian *UserControl* (*ActiveX Controls*) [4] :

1. Penulisan *userControl* untuk mendapatkan efek khusus
2. Penghalusan (*Enhance*) kontrol yang telah ada
3. Penyusunan ulang beberapa kontrol yang telah ada menjadi *userControl*

Metode ini sangat cocok dipakai untuk template dengan menambah customisasi suatu obyek, namun kendalanya adalah bisa jadi obyek yang dipakai banyak dengan spesifikasi tertentu, sehingga diperlukan *userControl* yang banyak pula. Jika ditemukan obyek baru, maka program akan dibuka ulang untuk menambah *userControl* obyek tersebut.

Penambahan *userControl* dapat ditambahkan sebagai kompilasi file pustaka (*.OCX) , yang dapat dipakai sebagai referensi pustaka dengan lisensi dari program yang saat itu jalan atau referensi dari file hasil kompilasi pustaka (*.OCX)

3. METODOLOGI

Penggunaan *template form* dan penambahan kontrol secara dinamis harus dapat ditemukan kembali sehingga *event* yang terjadi tidak terlepas. Ada dua macam metode yang dipakai saat ini :

1. Sistem hook dengan berbasis titik koordinat murni
2. Sistem hook dengan kombinasi titik koordinat dan pelacakan dari koleksi kelas yang terbentuk

3.1. Sistem hook dengan berbasis titik koordinat murni

Metode ini dikemukakan oleh Steve McMahon [5] dengan memakai kontrol yang telah ada sebelumnya.

Form dibuat dengan kontrol yang tidak ditambahkan secara dinamis melainkan saat *design-time* di tambahkan. Kemudia dicari nama dari kontrol tersebut, untuk diberi nilai *property*

Algoritma :

1. Pada *design-time*, buat form dan tambahkan semua kontrol yang ada
2. *Load* kontrol window (vbahook.dll)
3. Program ulang form dengan merefer ke vbahook.dll
4. Install Hook ke form yang telah dibuat

Contoh algoritma pelacakan kontrol yang dikemukakan oleh Steve McMahon

1. Alih Sandi (*Decode*) parameter window saat terkena hook yaitu lparam dan wparam
2. *CopyMemory* dari lParam ke tipe data *struct* berupa *mouse handling*
3. Ambil koordinat Form berdasarkan dari POINT API yang telah tercopi

   WindowFromPoint(x, y)
4. Jika tidak berhasil (nilai = 0) kerjakan nomor 1 lagi
5. Buat variabel pemegang kontrol dan operasikan ke semua kontrol form

6. Untuk masing-maisng Kontrol di form, cek apakah memiliki hWnd (*handle window*) yang sama dengan hwnd hasil nomor 2
7. Jika sama berarti kontrol telah ditemukan, jika tidak lanjutkan ke nomor 8
8. Ambil obyek dan temukan childnya dengan fungsi EnumChildWindows
9. Jika ditemukan, ulangi nomor 6 dengan parameter hWnd dengan *container* dari *child* obyek tersebut
10. Jika tidak ditemukan bandingkan posisi koordinat dengan refer rumus (a2)
11. Jika koordinat dari nomor 2 melingkupi koordinat obyek N1 <= 1 dan N2 <= 1, maka obyek ditemukan

Lingkup obyek :

```
Public Type MOUSEHOOKSTRUCT
   pt As POINTAPI
   hWnd As Long
   wHitTestCode As Long
   dwExtraInfo As Long
End Type
Private m_tMHS As MOUSEHOOKSTRUCT
CopyMemory m_tMHS, ByVal lParam, Len(m_tMHS)
Private Type POINTAPI
  x As Long
  y As Long
End Type
Dim tP As POINTAPI
tP.x = x: tP.y = y
ScreenToClient objParent.hwnd, tP
```

```
P0 = ctl.Left \ Screen.TwipsPerPixelX
P1 = (ctl.Left + ctl.Width) \ Screen.TwipsPerPixelX
P2 = ctl.Top \ Screen.TwipsPerPixelX
P3 = (ctl.Top + ctl.Height) \ Screen.TwipsPerPixelX
```

…… (a1)

```
N1 = tp.x /(P1-P0 )
N2 = tp.y/(P3-P2)
```

…… (a2)

Setelah melakukan ujicoba semua kontrol yang menjadi sampel dapat ditemukan termasuk *child* yang bergantung pada formnya, bahkan untuk obyek yang tidak memiliki hWnd dapat ditemukan seperti misalnya kontrol label.

Saat dicoba untuk menambahkan menu baik *pulldown* menu maupun *popup* menu, maka semua nama kontrol menjadi kacau, semua merefer ke nama kontrol menu

Kesimpulan metode sistem hook dengan berbasis koordinat murni memiliki keterbatasan saat form harus memuat menu-menu yang dihandelnya

3.2 Sistem hook dengan kombinasi titik koordinat dan pelacakan dari koleksi kelas yang terbentuk

Kelemahan metode pertama menjadi titik fokus dari penelitian ini, sebab keperluan pelacakan kembali menjadi titik tolak kelanjutan visualisasi obyek berdasarkan model HTA.

Pendekatan ulang sebagai perbedaan kedua metode dapat dilihat pada tabel berikut ini

Tabel 1. Perbandingan Metode

Pendekatan masalah	Koordinat murni	Kombinasi koordinat
Pembuatan obyek	design-time	run-time
Kontrol Obyek	Form	Modul
Pemakaian Kelas	Proses decode	Identfikasi obyek
Menu	Tidak terhandel	Terhandel
Koleksi Kelas	Tidak diperlukan	Diperlukan
Kode Program di Form	Harus ada	Otomatis terhandel
Aturan kontrol	Dari kode program	Dari aturan database

Pembuatan obyek secara dinamis mengakibatkan kehilangan kontrol dan hanya bertumpu pada urutan kerja yang dihandel oleh modul. Form yang ada hanya bersifat penampung kontrol-kontrol saja, sedangkan *property, method* dan *event* semua di pegang oleh modul yang merefer ke database

Semua kontrol, baik form, menu maupun *kontrol intrinsic* ataupun *kontrol extender* dapat dibuat berdasarkan inisial yang telah ada sebagai aturan di database. Kontrol yang dibuat disimpan didalam kelas sebagai identifikasi obyek. Kelas-kelas penyimpanan identifikasi obyek antara lain :

1 Kelas FormID
2 Kelas MenuID
3 Kelas ObjectID

Setiap obyek yang dibuat berupa instan dari kelas, maka disimpan identitasnya, terutama kontrol hWnd untuk mengetahui *parent, child* dan *container*-nya. Pelacakan kembali dapat melalui hWnd yang telah tersimpan saat *event* terjadi misalnya tombol keyboard atau mouse ditekan

Kontrol yang dibuat memiliki karakteristik :

1. memiliki hWnd sendiri
2. hWnd ikut dengan form
3. tidak memiliki sama sekali kontrol hWnd

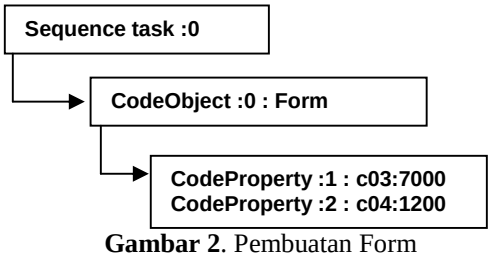
Berdasarkan karakteristik kontrol tersbut, maka dapat dibagi obyek berdasarkan perlakuan obyek. Pembagian obyek secara garis besar untuk memisahkan perlakuan obyek adalah :

1. Form
2. Menu
3. Kontrol intrinsic
4. Kontrol Extender

3.2.1 Form

Obyek form tidak dapat dibuat sebagai instan kelas dari form yang belum ada, sehingga diperlukan satu form kosong template sebagai pancingan tanpa harus ada kode program di form tersebut.

Form yang dibuat berdasarkan kode aturan yang tersimpan ke database dan kode tersebut disimpan sebagai identitas form yang tersimpan dalam kelas FormID, contoh pembuatan form.



Gambar 2. Pembuatan Form

Urutan pembuatan menjelaskan *parent* dari form tersebut., sehingga suatu form dapat memiliki form lain sebagai *child* - nya.

Tabel 2. Pustaka Property

Type	PropertyName	PcallType
c03	**Height**	**4**
c04	**Width**	**4**

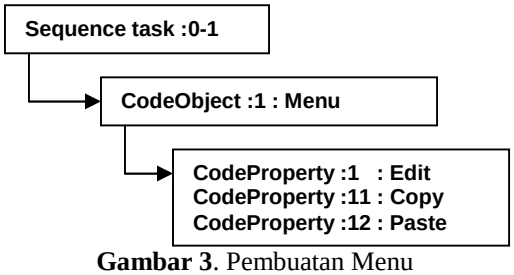
Tabel 2 dipakai untuk referensi kode-kode *property* dari tugas yang dikerjakan, sedangakan bagaimana mengerjakannya tergantung dari tipe panggilan seperti tabel 3 berikut ini.

Tabel 3. Pustaka CallType

TypeCall	DescriptionCallType
1	**VbMethod**
2	**VbGet**
4	**VbLet**
8	**VbSet**
10	**Ass Value**

3.2.2 Menu

Pembuatan menu dapat dilakukan jika sudah ada formnya, sehingga secara hirarki tugas, Menu menjadi subtugas seperti ditunjukkan pada gambar 3.



Gambar 3. Pembuatan Menu

Pada level *parent* dipakai fungsi API (*Aplication Programming Interface*) untuk mendapatakan *parent* hWnd:

HWndParent = CreateMenu()

Sedangkan untuk submenu memakai harus mencari *parent* hWnd yang telah ada dalam kelas MenuID. Kemudian perintah pembuatan menu dapat dilakukan dengan fungsi

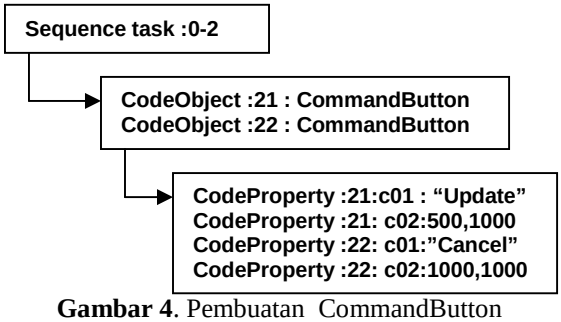
InsertMenu .Parent, -1, wFlags, .MenuId, .Caption

Setelah menu terbentuk maka untuk menjaga indentitas dari menu-menu, semua informasi tentang menu harus disimpan dalam MenuID, terutama hWnd yang menjadi dasar hirarki menu

3.2.3 Kontrol Intrinsic

Kontrol *intrinsic* adalah kontrol standar yang tersedia secara *default*. Kontrol tersebut tersimpan dalam pustaka VB, sehingga semua kelas yang ada merefer ke pustakaVB.

Syarat pembuatan kontrol *intrinsic* adalah harus sudah ada formnya dan bisa secara bersamaan ada dengan kontrol lain atau menu. Hubungan ini bisa digambarkan sebagai kesejajaran atau *optional task.*



Gambar 4. Pembuatan CommandButton

CodeProperty untuk c02 adalah *move* yaitu suatu *method* dari obyek yang ada dengan mengatur koordinat perpindahan obyek.

Semua obyek (dalam kasus ini ada dua buah obyek) akan tersimpam identasnya di kelas ControlID, terutama hWnd. Biasanya untuk kontrol

intrinsic memiliki kontrol hWnd sendiri kecuali beberapa kontrol seperti label

3.2.4 Kontrol Extender

Berbagai macam kontrol muncul sesuai dengan perkembangan yang ada. Banyak kontrol *intrinsic* mengalami perubahan dan banyak pula kontrol-kontrol baru muncul. Kontrol yang muncul tetap memiliki pola yang sama terhadap kebutuhan obyek yaitu *property*, *event* dan *method*

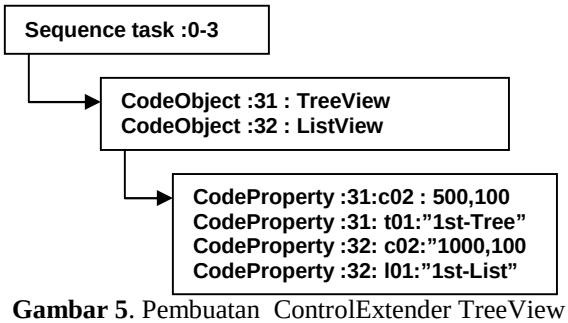
Contoh kontrol yang mengalami perubahan :

Kontrol Data	-> ADODC
List	->DBList/DataList
Combo	->DBCombo

Contoh kontrol baru adalah :

Dtpicker, ImageList,TreeView

Kontrol *extender* banyak yang tidak memiliki kontrol hWnd, sebab hWnd ikut dalam form parent-nya, sedangkan didalam kontrol tersebut dapat diterjemahkan sebagai parameter Lparam dari fungsi Hook yang memegang semua kelakuan dari form yang di beri Hook.



Gambar 5. Pembuatan ControlExtender TreeView & ListView

Contoh pada gambar 5 terdapat *method* baru yang direferensi yaitu :

t01: Object.Nodes.Add
l01: Object.ListItems.Add

Semua kontrol obyek *extender* dapat dimasukkan sebagai aturan baik berupa *property*, *method* ataupun bahkan *event* yang akan dikerjakan.

Program akan mendeteksi *event* yang berlaku umum untuk semua obyek misal :

- Event Click
- Event DbClick
- Event KeyUp dsb

3.2.5 Deteksi Event

Di dalam obyek *event* yang terjadi pada semua obyek tersimpan dalam aturan database dengan parameter aturan yang perlu dituliskan :

1. Type *event*
2. Obyek penampung hasil *event*
3. Obyek penggerak *event*
4. Kunci pencarian jika ada sub item
5. Koneksi database jika ada (bisa SQL,Oracle atau yang lain tergantung dari *method open* dengan *property ConnectionString*
6. Urutan pengerjaan jika terdapat beberapa tugas yang harus dikerjakan

Beberapa *event* dibangkitkan oleh aksi eksternal yang dapat berupa pergerakan mouse atau keyboard maupun event program lain. Semua pergerakan mouse, keyboard maupun event lain yang timbul dikirim oleh sistem operasi Windows melalui *sendMessage* sebagai IPC (*interProcess Communication*). Kita bisa mengalihkan (*intercept*) aksi tersebut yang kemudian dibelokkan ke handel program kita.

Pada penelitan ini dipakai fungsi API :

```
procOld = SetWindowLong(hwnd, GWL_WNDPROC,
                        AddressOf newWindow)
```

ProcOld	: handel program sebelum dialihkan
newWindow	: fungsi baru untuk intersep program
hWnd	: Kode window yang di alihkan

Terakhir setelah dialihkan, maka window harus dikembalikan pada program semula dengan fungsi API :

```
tmp = SetProp(hwnd, hwnd, procOld)
```

Semua *event* yang terjadi akan ditangani oleh fungsi baru hasil buatan sendiri yaitu newWindow() yang secara default memberikan nilai paramter :

```
(ByVal hwnd As Long, ByVal info As Long, _
 ByVal wParam As Long, ByVal lParam As Long)
```

HWnd	: kode handel window
Info	: informasi event dengan kode tertentu
Wparam	: informasi obyek
Lparam	: informasi handel obyek

Dari hasil parameter tersebut dapat diolah lagi menjadi pelacakan kembali obyek-obyek yang telah dibuat secara dinamis pada waktu sebelumnya

3.2.6 Deteksi Obyek

Pendekatan deteksi obyek ini berbeda dengan metode pertama seperti diperlihatkan pada tabel 1. Obyek yang dibuat secara dinamis lewat aturan HTA dapat dilacak keberadaannya dengan mengambil informasi pada kelas-kelas yang dibuat.

Alur logika penjejakan kembali obyek yang ada adalah :

1. Ambil informasi dari deteksi *event*
2. Berdasarkan parameter info, buat klasifikasi kode
   - &H111
     telah terjadi *event* mouse kontrol terlepas (*independent*)
   - 528 & wParam=513
     klik kiri *event* mouse kontrol bergantung (*dependent*)
   - 528 & wParam=516 : klik kanan
     klik kanan *event* mouse kontrol bergantung (*dependent)*
3. Untuk kode &H111, ambil Wparam dan Lparam untuk menentukan kunci dari koleksi kelas ObjectID, jika ditemukan berarti obyek ambil namaObyek
4. Jika tidak ditemukan ulangi nomor 3 dengan Koleksi kelas pada MenuID
5. Untuk kode klik kiri dan kanan ambil parameter hwnd, lparam dan wparam
6. Cek untuk masing-masing kontrol yang ada di form yang aktif , lakukan perbandingan kontrol hwnd dengan parameter hwnd
7. jika perbandingan sama, maka cek apakah nama kontrol merupakan nama parent menu
8. jika tidak, ambil posisi kursor saat ini
   GetCursorPos Tpoint
9. Lakukan perhitungan dengan rumus (b1) & (b2)
10. Jika N1 <= 1 dan N2 <=1 , maka kontrol telah ditemukan

    N1 = (x – (XL + XF)) / WL ..... (b1)
    N2 = (y – (YL+YF)) / HL .....(b2)

    Dimana :
    X : koordinat posisi x saat ini
    Y : koordinat posisi y saat ini
    XL : posisi x kontrol
    XY : posisi y kontrol
    XF : posisi x Form Aktif
    YF : posisi y form aktif
    WL : lebar kontrol
    HL : Tinggi kontrol

4. UJI COBA

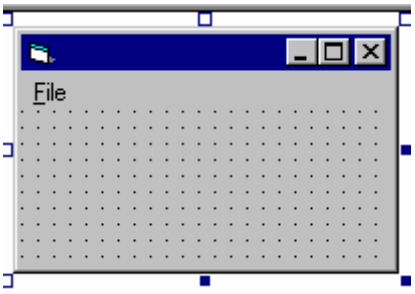
Data sampel yang digunakan dalam uji coba ini terdiri dari beberapa kontrol yang mewakili sebagian besar kontrol-kontrol yang ada di visual basic dan memiliki *event* tertentu

Kontrol yang dibuat dapat dilhat ditabel 4 berikut ini :

Tabel 4. Sampel Kontrol untuk uji coba

Kode Kontrol	Anggota Pustaka	Jumlah	Tipe kontrol
0000		1	FORM
0001		4	Menu
1000	VB	3	CommandButton
2001	MSComctlLib	1	ListView
2002	MSComctlLib	1	TreeView

Pertama kali akan terlihat form kosong tanpa kontrol seperti pada gambar 5.



Gambar 5. Form template kosong.

Setelah program dijalankan maka terlihat Form dengan semua kontrol yang telah terdefinisi dengan paramater yang dimasukkan seperti di tabel 5 berikut ini.

tblPropertiesObject : Table

Prop	PropLe	Prop	PTyp	PropertyValue
0	000	1	c03	3000
0	000	2	c04	5000
1	000-001	1	c03	1500
1	000-001	2	c04	2500
1	000-001	3	t01	,,"P0" , "MENU UTAMA PIKTI-ITS"
1	000-001	9	t02	True
2	000-001	1	c03	1500
2	000-001	2	c02	2500
2	000-001	3	c04	2500
2	000-001	4	t02	True
2	000-001	5	l02	3
2	000-001	6	l03	True
3	000-001	1	c01	&Update
3	000-001	2	c02	10,1550
3	000-002	1	c01	&Cancel
3	000-002	2	c02	1210,1550
3	000-003	1	c01	&Tutup
3	000-003	2	c02	2410,1550
5	000	1	c10	PROVIDER=MSDataShape;Data PROVIDER=MSI
0		0		

Record: 17 of 19

Tabel 5. Tabel Sampel Property dan Method untuk uji coba.

Tabel Property diatas merefer ke dalam type property ataupun method seperti tabel 6 berikut ini.

Tabel 6. Sampel Type Of Property.

TypeOfProperty : Table

	TypePropertyI	PropertyName	PCallType
▶	c01	Caption	4
	c02	Move	1
	c03	Height	4
	c04	Width	4
	c10	ConnectionString	10
	l00	Object.ListItems.Clear	1
	l01	Object.ListItems.Add	1
	l02	Object.View	10
	l03	Object.GridLines	10
	t01	Object.Nodes.Add	1
	t02	Object.HotTracking	10

Setting menu untuk ujicoba kali ini seperti pada tabel 7 berikut ini.

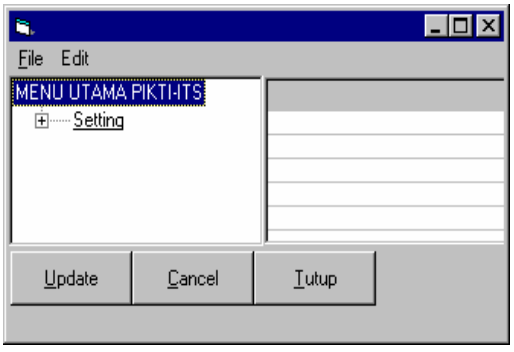
Tabel 7. Sampel Menu yang ditampilkan.

tblSubTaskDesc : Table

	DSeq	DLevel	DHierarchic	Descrip
▶	4	000-001	000-001	Edit
	4	000-001	000-001-001	Cut
	4	000-001	000-001-002	Paste
	4	000-001	000-001-003	-
	4	000-001	000-001-004	Find
*	0			

Record: 1 of 5

Program dijalankan akan terlihat seperti gambar 6 berikut ini :

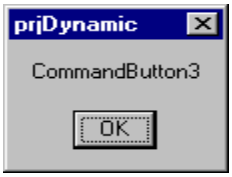


Gambar 6. Hasil Pembuatan kontrol secara dinamis.

Setelah semua setting dilakukan untuk ujicoba, maka program dijalankan untuk deteksi obyek yang terkena event dari mouse.

Dengan event klik masing-masing obyek seperti terlihat sebagai berikut ini :

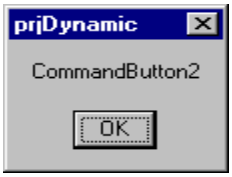
1. Hasil pelacakan kembali nama obyek tombol "Tutup" saat ditekan tombol mouse (click)



Gambar 7. Hasil Tombol Tutup

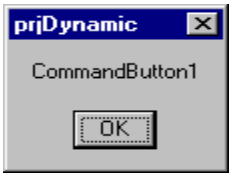
2. Hasil pelacakan kembali nama obyek tombol "Cancel" saat ditekan tombol mouse (click)

Gambar 8. Hasil Tombol Cancel



3. Hasil pelacakan kembali nama obyek tombol "Update" ditekan tombol mouse (*click*)

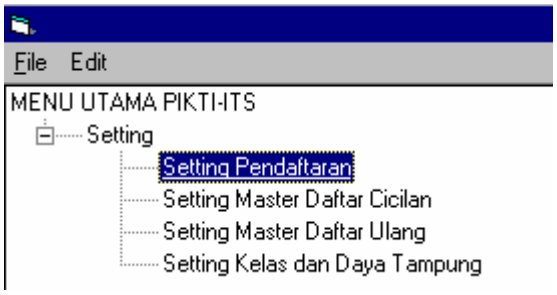
Gambar 9. Hasil Tombol Update



4. Hasil pelacakan kembali nama obyek menu Edit dengan sub menu "Cut" dan ditekan tombol mouse (*click*)

Gambar 10. Hasil Tombol Update

5. Hasil pelacakan kembali nama obyek TreeView dan ditekan tombol mouse (click)



Gambar 11. Hasil TreeView di tekan tombol Mouse

Hasil dari pelacakan obyek TreeView kemudian dimasukkan ke dalam obyek lain yaitu ListView

S...	SPD_Ua...	SPD_noPendaftaran	SP...
9900	50000	V/99/PIKTI/	Ketua
0001	60000	VI/00/PIKTI/	Ketua
9899	40000	IV/98/PIKTI/	Ketua
0102	75000	VII/01/PIKTI/	Ketua
0203	80000	VIII/02/PIKTI/	Ketua
0209	60000	V/99/PIKTI/	Ketua
0304	85000	IX/03/PIKTI/	adm01

Gambar 12. Hasil ListView dari event click TreeView

5. KESIMPULAN

Dari hasil uji coba yang telah dilakukan dan perbandingan dua metode pelacakan obyek, didapatkan beberapa kesimpulan sebagai berikut :

1. Penggunaan IPC antar window dapat dimanfaat untuk memanipulasi parameter untuk deteksi obyek
2. Obyek yang tertampung dalam window harus dijaga hirarkinya selain mememanfaatkan parameter hWnd, juga perlu disimpan identitas lain untuk menghindari obyek yang sama terdeteksi lain. Pada kasus metode pertama mengalami kegagalan untuk penambahan obyek menu, dengan pendekatan pembuatan obyek secara dinamis pada run-time kegagalan tersebut mampu di atasi
3. Selama melakukan ujicoba juga diperlihatkan secara natural kecepatan loading data ke memori terutama aturan-aturan yang tersimpan dalam database, secara natural pembukaan database diawal program yang mempengaruhi kecepatan program, namun setelah semua terload, maka bisa diambil logika berupa pembuatan obyek sama halnya dengan pembuatan instan kelas pada konsep OOP (*object oriented programming*)

DAFTAR PUSTAKA

[1] J. Annet dan K.D. Duncan *Task Analysis and training design.*, Occuppational Psychology,41,1967

[2] Alan Dix, Jannet Finlay, Gregory Abowd dan Russel Beale, *Human Computer Interaction*, Prenctice Hall,1993.

[3] David McMahon, *Rapid Aplication With Visual Basic 6*, McGraw-Hill,2000

[4] Guy Eddon & Henry Eddon, *InProgramming Components With Microsoft Visual Basic 6.0*, Microfot Press, 1998

[5] Steve McMcMahon, *VBHelper : journal*,1998