BAB II

LANDASAN TEORI

II.1 Teknologi .NET

Microsoft .NET ialah sebuah platform untuk membangun, menjalankan dan meningkatkan generasi lanjut dari aplikasi terdistribusi. Microsoft .NET memperluas client, server dan service-service yang terdiri atas:

1) Sebuah model pemrograman yang memungkinkan developer membangun Web services dan aplikasi.
2) Sekumpulan XML Web services seperti Microsoft .NET My Services, yang membantu developer menghasilkan aplikasi yang simple dan terpadu.
3) Sekumpulan server, termasuk Windows 2000, SQL Server, dan BizTalk Server, yang memadukan, menjalankan dan mengoperasikan serta menangani XML Web Services dan aplikasi.
4) Tool seperti Visual Studio.NET untuk membangun XML Web Service dan aplikasi untuk window dan web.
5) Piranti lunak klien, seperti Windows XP dan Windows CE.

II.1.1. NET Framework

Microsoft .NET ialah model pemrograman dari platform .NET untuk membangun, menyebarkan dan menjalankan XML Web Service dan aplikasi. .NET Framework menyediakan lingkungan berbasis standar produksi yang tinggi untuk memadukan investasi yang ada dengan aplikasi dan service generasi yang akan datang. .NET Framework terdiri atas dua bagian utama, yaitu CLR dan gabungan kelas library termasuk ASP.NET untuk

aplikasi web dan XML Web Services, Windows forms untuk aplikasi klien dan ADO.NET.

Visual Studio.NET dibangun menggunakan fondasi .NET Framework. .NET Framework menyediakan lingkungan yang cerdas, mudah dikembangkan untuk membangun, menyebarkan dan menjalankan XML Web Services yang terdistribusi serta aplikasi. Dalam istilah yang mudah, .NET memisahkan platform Sistem Operasi menjadi dua layer, yaitu sebuah layer pemrograman dan layer eksekusi.

II.2 User Interface Process Application Block

Aplikasi biasanya mempunyai kode yang mengatur interaksi dalam lapisan presentasi / tampilan. Misalnya dari suatu *form* kita menentukan *form* berikutnya yang dibuka oleh pengguna. Para *developer* biasanya menggabungkan kode untuk navigasi dengan *form* itu sendiri. Hal ini membuat kode program menjadi sangat kompleks dan sangat susah untuk di-*maintenance* dan dikembangkan lagi. Dalam kasus tertentu, sIstem informasi tidak dapat digabungkan dengan *platform* yang berbeda karena kontrol dan *state* aplikasi tidak dapat digunakan kembali. Dalam banyak kasus, sistem informasi memiliki *state* yang harus di-*maintenance*. Jika *state* disimpan di dalam *form* maka kode harus mengakses *form* untuk mendapatkan *state* tersebut. Hal ini agak sulit di implementasikan dan menghasilkan kode program yang tidak efisien. Sebagai contoh, jika kita mengembangkan beberapa *form* yang akan diakses berurutan, kemudian kita harus menambahkan *form* ditengah urutan tersebut

maka kita harus mengubah kode di *form* sebelum dan *form* sesudah urutan yang kita masukkan itu.

Ketika *user* menggunakan aplikasi, dia akan memulai sebuah *task*, kemudian ada kasus ketika dia harus berhenti dan kembali lagi. Jika user menutup aplikasi atau kehilangan *session*, maka ia harus mengulang kembali pekerjaan yang telah ia kerjakan dari awal. Untuk bisnis dalam skala besar tidak dapat diterima. Untuk itu, jika kita merancang sebuah aplikasi maka harus dipikirkan alur kerja, navigasi serta hubungan dengan layanan bisnis sebagai suatu bagian yang terpisah bagaimana data didapatkan dan bagaimana data disampaikan kepada pengguna.

User Interface Process (UIP) Application Block menyediakan *framework* yang memudahkan *developer* dalam memisahkan *bisnis logic* dengan *user interface*. Kita dapat menuliskan kode navigasi yang kompleks terpisah dari lapisan presentasi dan proses kerja yang dapat digunakan di berbagai alur bisnis dan serta dikembangkan sejalan perkembangan aplikasi. Dengan kata lain keuntungan utama dari *framework* ini adalah sistem navigasi yang lebih mudah digunakan dan sistem yang lebih *scalable*.

Ada beberapa konsep yang harus dimengerti sebelum mempelajari UIP yaitu:

1) *User Interface Process*

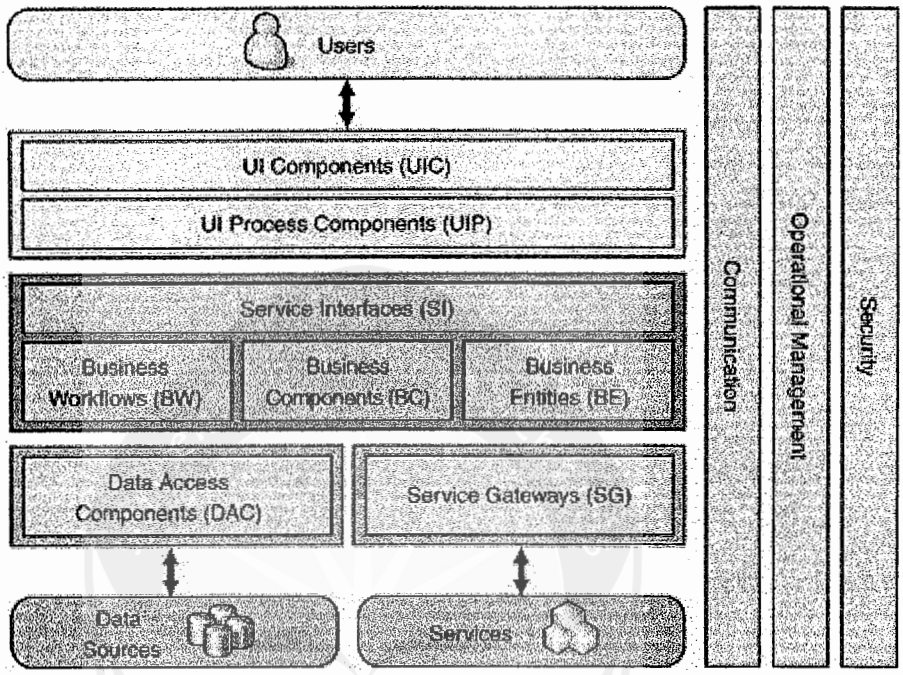
   Yaitu sekumpulan aktivitas *user* untuk mencapai tujuan. Dengan mengelompokkan, kita dapat mengetahui *view* yang digunakan untuk tiap proses dan bagaimana navigasi antar *view* tersebut.

2) *UIP Task*

Yaitu suatu *task* / proses yang sedang dikerjakan.

3) *State*

Yaitu potongan *task* / proses.



Gambar II.1 Lapisan aplikasi berdasarkan arsitektur aplikasi terdistribusi .Net

Diatas adalah gambar lapisan aplikasi berdasarkan arsitektur aplikasi terdistribusi .Net. UIP adalah *framework* untuk membangun *user interface process component*. Komponen ini bertanggung jawab untuk:

1) Menangani alur informasi dari *user interface component*.
2) Menangani transisi antar *stage* dari *user interface process*.
3) Mengubah *user process flow* jika terjadi kesalahan.
4) Membedakan *user interaction flow* dengan implementasi secara konseptual.

5) Me-*maintenance state internal* bisnis. Biasanya berhubungan dengan entitas yang dipengaruhi oleh *user interaction*.
6) Menjaga jalur task / proses pada *user interface process*.

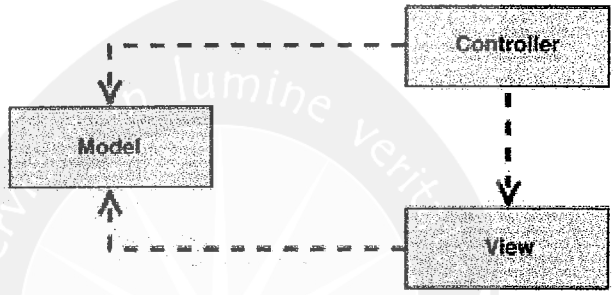
Dengan memisahkan *user interface process component* dengan *user interface component* kita dapat mengabstraksi kode yang mengatur keadaan aplikasi dari *user interface* itu sendiri. Ini memungkinkan kita untuk menuliskan kode yang dapat digunakan kembali untuk alur kontrol dan manajemen *state* pada tipe aplikasi yang berbeda-beda, tergantung kegunaan *user interface*. Hal ini juga memungkinkan kita untuk mengganti *user interface* dan mengatur berbagai user *interface* tanpa mengganti banyak kode dan memberi kita lebih banyak fleksibilitas dalam mencoba aplikasi kita karena kita dapat membagi prosedur pencobaan.

II.1.2. Model – View - Controller

UIP Application Block berdasarkan pada pola *model-view-controller* (*MVC*). Ini adalah pola untuk memisahkan *user interface logic* dari *bisnis logic*. Pola ini memisahkan model *domain application*, *application presentation*, dan aksi berdasarkan *user input* dalam 3 hal berikut:

1) *Model - Object* ini mengetahui semua tentang data yang akan ditampilkan dan bertanggung jawab untuk mengatur data dan *action* dari aplikasi. Ini dapat diartikan sebagai bagian *process* pada sistem *input-process-output*.

2) *View* - *Object* ini mengatur informasi yang ditampilkan kepada user. Berbagai *view* dapat digunakan untuk menampilkan data kepada *user* dengan berbagai cara. Ini dapat diartikan sebagai *output*.
3) *Controller* - *Object* ini memampukan *user* untuk berinteraksi dengan aplikasi. Ini membutuhkan *input* dari user dan mengirimkan instruki ke *Model*. Ini dapat diartikan sebagai *input*.



Gambar II.2 MVC Class Structure

View dan *Controller* keduanya bergantung pada *Model*. Bagaimanapun, *Model* juga tergantung pada *View* dan *Controller*. Ini adalah salah satu keuntungan dari pemisahan. Pemisahan ini memampukan *Model* untuk dibuat dan diuji secara terpisah dari presentasi visual.

Tepatnya dalam menggunakan *MVC* tergantung dari tipe aplikasi yang kita buat. Sebagai contoh, pemisahan antara *View* dan *Controller* tidak begitu diperhatikan dalam banyak aplikasi *client server*, dan banyak pengguna *framework user interface* menyatukan sebagai sebuah *object*. Sedangkan pada aplikasi *web*, pemisahan antara *View*(*browser*) dan *Controller*(*server-side component*) dibatasi dengan sangat baik.

UIP Application Block dirancang untuk membantu kita dalam:

1) Mengabstraksi semua navigasi dan *workflow* dari *user interface*.
2) Mampu menggunakan model programming yang sama dalam tipe aplikasi yang berbeda.
3) Menghilangkan semua manajemen *state* dalam *user interface*.
4) Mengetahui *state* tertentu diantara *process*.

Workflow dari aplikasi adalah proses bisnis, bukan proses *user interface* dan pengatur nya harus berada di luar lapisan *user interface*. Abstraksi kode juga memudahkan kita untuk me-*maintainance* dan menambahkan aplikasi yang ada dengan lebih mudah. *UIP Application Block* memisahkan kode *workflow* dari lapisan *user interface* ke lapisan *user interface proses*. Dimana nvigasi dari *worfklow* tidak di "*hardcode*" dalam *user interface*, maka jika *flow* dari aplikasi berubah maka kita hanya butuh mengubah *workflow* nya, tidak termasuk elemen dalam proses tersebut.

Kita mungkin saja membuat aplikasi berbasis *windows* untuk penggunaan *internal*, dan suatu saat nanti butuh mengembangkan aplikasi berbasis *web* untuk penggunaan *eksternal*. Jika mungkin, juga mengembangkan *web* untuk di akses oleh alat yang berbeda. Jika kita menggunakan model pemrograman yang sama untuk semua tipe aplikasi maka akan lebih mudah mengintegrasikan aplikasi dari *platform* yang satu ke *platform* yang lain.

Dalam banyak aplikasi, *state* disimpan dalam *user interface*. Ini berarti bahwa kode yang rumit dibutuhkan untuk menyimpan, mengakses, dan mengubah *state*

tersebut. Sebagai tambahan *state* yang ada tergantung pada tipe *user interface*, yang akan menyusahkan jika menggunakan tipe aplikasi yang berbeda. *UIP Navigation Block* memisahkan *state* dari *user interface* dan menyimpan di *object* umum yang dapat diakses dengan *class block*. Ini berarti suatu tampilan dapat bekerja dengan suatu *state* tanpa harus mengetahui dari mana *state* tersebut berasal atau siapa yang membuatnya.

Dalam banyak aplikasi juga terdapat kesulitan untuk menyimpan *state* dari suatu proses pada suatu saat tertentu. Banyak aplikasi menggunakan transaksi untuk membungkus suatu proses tapi hanya memperbolehkan pengaksesan *state* sampai transaksi itu berhasil. Ini berarti jika sistem berhenti atau *user* meninggalkan *website* maka akan sulit untuk kembali ke keadaan sesaat sebelum dia meninggalkan proses tersebut. *UIP Application Block* mencakup *state persistence* mekanisme yang memampukan kita untuk mengetahui dengan pasti *state* proses. Ini memampukan kita untuk mengembangkan aplikasi berdasarkan *state persistence* pada *interval* yang telah ditentukan, dengan artian kita dapat mengembalikan keadaan aplikasi pada suatu saat nanti. Selain menyimpan di memori aplikasi, kita juga dapat menyimpan di media penyimpanan ataupun data base untuk penggunaan kembali nantinya.

II.1.3. UIP Terminology

II.1.3.1. Configuration File

UIP Application Block menggunakan *file application configuration* untuk mengetahui tipe *object* yang harus diketahui, *user interface proses* apa saja yang ada pada aplikasi, *navigator* apa yang digunakan, *shared transition* apa yang ada, dan *view* apa yang ada. *UIP configuration* dibungkus dengan *section application configuration* yang dinamakan *<uipConfiguration>*.

File yang digunakan berbasis XML, yang memegang peranan penting dalam proses *user interface* aplikasi, *view* yang digunakan, dan *navigation* yang menentukan jalannya *workflow* aplikasi. Isi dari setiap file biasanya sama. Untuk aplikasi berbasis *windows*, kita menggunakan *app.config* sedangkan untuk aplikasi berbasis *web*, kita menggunakan *web.config*.

Pertama kita harus membuat sebuah *configuration section* untuk menentukan setting *UIP* kita. *Section* ini harus dituliskan di dalam *<configSections>*.

```
<configuration>
        <configSections>
        <section name="uipConfiguration"
        type="Microsoft.ApplicationBlocks.UIProcess.UIPConfigHandler,
        Microsoft.ApplicationBlocks.UIProcess,
        Version=1.0.1.0,Culture=neutral,PublicKeyToken=null"/>
        </configSections>
</configuration>
```

Setelah itu kita harus menambahkan *<uipConfiguration>* bagian itu sendiri. Bagian ini digunakan untuk menentukan *class* yang digunakan untuk *view* manajemen aplikasi, *state management*, *controller*, *view* yang digunakan oleh *user*, dan *navigator* yang menentukan *workflow* antar *view* yang ada. Struktur dari bagian *<uipConfiguration>* adalah seperti berikut:

```
<uipConfiguration>
        <objectTypes>
        . . .
        </objectTypes>
        <views>
        . . .
        </views>
        <sharedTransitions>
        . . .
        </sharedTransitions>
        <navigationGraph>
        . . .
        </navigationGraph>
        <uipWizard>
        . . .
        </uipWizard>
        <userControls>
        . . .
        </userControls>
</uipConfiguration>
```

1) *objectTypes* : menentukan *class* yang digunakan untuk mengatur dan menentukan *user interface* dalam aplikasi. *objectTypes* sendiri mempuyai attribut :

Tabel II.1 Tabel atribut objectTypes

name	nama yang digunakan untuk *object* tersebut dalam *configuration file*.
type	tipe dari *object* tersebut yang menetukan *assembly* dimana *object* tersebut berada dan nama *class* nya.

Bagian *objectTypes* ini terdiri dari:

```
<objectTypes>
        <iViewManager . . . />
        <state . . . />
        <controller . . . />
        <layoutManager . . . />
        <statePersistenceProvider . . . />
```

```
</objectTypes>
```

Tabel II.2 Tabel bagian objectTypes

iViewManager	berisi view manager yang bertanggung jawab dalam mengaktifkan dan menonaktifkan view dalam aplikasi.
state	mengatur state dari aplikasi.
cotroller	mengatur flow dari view dan respond dari request dari view.
layoutManager (optional)	menampilkan custom layout untuk controller dan view.
statePersistenceProvider	menentukan lokasi dimana state antar transisi harus disimpan

Contoh kode *<objectTypes>* aplikasi *windows*

```
<objectTypes>
        <iViewManager name="WindowsFormViewManager"
        type="Microsoft.ApplicationBlocks.UIProcess.WindowsFormViewManager,
        Microsoft.ApplicationBlocks.UIProcess,
        Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"/>
        <state name="State"
        type="Microsoft.ApplicationBlocks.UIProcess.State,
        Microsoft.ApplicationBlocks.UIProcess,
        Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"/>
        <controller name="MyController" type="Test.MyController, Test,
        Version=1.0.0.0,Culture=neutral,PublicKeyToken=null"/>
        <layoutManager name="HorizontalLayoutManager"
        type="LayoutManager.HorizontalLayoutManager, LayoutManager,
        Version=1.0.1.0,Culture=neutral, PublicKeyToken=null" />
        <layoutManager name="VerticalLayoutManager"
        type="LayoutManager.VerticalLayoutManager, LayoutManager
        Version=1.0.1.0,Culture=neutral ,PublicKeyToken=null" />
```

```
        <statePersistenceProvider name="SqlServerPersistState"
        type="Microsoft.ApplicationBlocks.UIProcess.SqlServerPersistState,M
        icrosoft.ApplicationBlocks.UIProcess, Version=1.0.0.0,
        Culture=neutral, PublicKeyToken=null"
        connectionString="Data Source=localhost;Initial
        Catalog=UIPState;Trusted_Connection=True"/>
    </objectTypes>
```

Contoh kode *<objectTypes>* aplikasi *web*

```
    <objectTypes>
        <iViewManager name="WebFormViewManager"
        type="Microsoft.ApplicationBlocks.UIProcess.WindowsFormViewManager,
        Microsoft.ApplicationBlocks.UIProcess,
        Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"/>
        <state name="State"
        type="Microsoft.ApplicationBlocks.UIProcess.State,
        Microsoft.ApplicationBlocks.UIProcess,
        Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"/>
        <controller name="MyController" type="Test.MyController, Test,
        Version=1.0.0.0,Culture=neutral,PublicKeyToken=null"/>
        <layoutManager name="HorizontalLayoutManager"
        type="LayoutManager.HorizontalLayoutManager, LayoutManager,
        Version=1.0.1.0,Culture=neutral, PublicKeyToken=null" />
        <layoutManager name="VerticalLayoutManager"
        type="LayoutManager.VerticalLayoutManager, LayoutManager
        Version=1.0.1.0,Culture=neutral ,PublicKeyToken=null" />
        <statePersistenceProvider name="SqlServerPersistState"
        type="Microsoft.ApplicationBlocks.UIProcess.SqlServerPersistState,M
        icrosoft.ApplicationBlocks.UIProcess, Version=1.0.0.0,
        Culture=neutral, PublicKeyToken=null"
        connectionString="Data Source=localhost;Initial
        Catalog=UIPState;Trusted_Connection=True"/>
    </objectTypes>
```

2) *views* : menentukan detail dari *view* yang digunakan dalam aplikasi. *views* mempunyai attribut :

Tabel II.3 Tabel atribut views

name	nama dari *view*.
type	class dari *view*.
controller	*controller* yang di deklarasikan di bagian

	<objectType>.
layoutManager (optional)	nama *layout manager.*
stayOpen (windows)	menentukan apakah *view* tetap terbuka ketika *user* pergi dari *view* tersebut.
openModal (windows)	menentukan apakah *view* terbuka sebagai *modal view.*
canHaveFloatingWindows (windows)	menentukan apakah *view* dapat mempunyai *floating windows.*
Floatable (windows)	menentukan apakah *view* harus tampil *float.*
any (optional)	*custom element & attribut.*

Contoh kode *<views>* aplikasi *windows*

```
<views>
        <view name="Form1" type="Demo.Form1, Demo, Version=1.0.0.0,
        Culture=neutral, PublicKeyToken=null" controller="MyController"
        layoutManager="VerticalLayoutManager" stayOpen="true"/>
        <view name="Form2" type="Demo.Form2, Demo, Version=1.0.0.0,
        Culture=neutral, PublicKeyToken=null" controller="MyController" />
        <view name="Form3" type="Demo.Form3, Demo, Version=1.0.0.0,
        Culture=neutral, PublicKeyToken=null" controller="MyController" />
        <view name="Form4" type="Demo.Form4, Demo, Version=1.0.0.0,
        Culture=neutral, PublicKeyToken=null" controller="MyController" />
        <view name="Error" type="Demo.Error, Demo, Version=1.0.0.0,
        Culture=neutral, PublicKeyToken=null" controller="MyController"
        stayOpen=false openModal=true />
</views>
```

Contoh kode *<views>* aplikasi *web*

```
<views>
        <view name="Page1" type="Page1.aspx" controller="MyController" />
        <view name="Page2" type="Page2.aspx" controller="MyController" />
        <view name="Page3" type="Page3.aspx" controller="MyController" />
        <view name="Page4" type="Page4.aspx" controller="MyController" />
        <view name="Error" type="Error.aspx" controller="MyController" />
</views>
```

3) *sharedTransitions* (*optional*) : menentukan *view* umum yang dapat diakses dari semua *view*.

Contoh kode <*sharedTransitions*>

```
<sharedTransitions>
      <sharedTransition navigateValue="storehelp" navigateTo='storehelp' />
</sharedTransitions>
```

4) *navigationGraph* : menentukan detail dari *graph navigator*. Dibutuhkan oleh tiap *task* yang menggunakan *graph navigator*. Digunakan untuk yang menggunakan percabangan alur *view*. *navigationGraph* mempunyai attribut:

Tabel II.4 Tabel atribut navigationGraph

startView	nama dari *view* pertama yang digunakan dalam proses. Ini tidak harus *view* pertama yang ditampilkan kepada pengguna.
iViewManager	nama *view manager* yang di deklarasikan di <*objectTypes*>.
name	nama dari *graph navigator*.
state	nama dari *state* yang di deklarasikan di <*objectTypes*>.
statePersist	nama dari *state persistence* yang di deklarasikan.
cacheExpirationMode *(optional)*	metode *expire* yang digunakan untuk menidak valid kan *cache*.
cacheExpirationInterval *(optional)*	waktu yang dibutuhkan untuk *expire*.

runInWizardMode (optional)	menentukan apakah *navigator* adalah *wizard navigator* untuk *window* aplikasi.
navigateValue	nilai yang diisikan untuk memicu rute atau jalan yang akan diambil.
view	*view* yang harus dibuka ketika nilai *navigateValue* digunakan.

Contoh kode *<navigationGraph>* aplikasi *windows*

```
<navigationGraph
        startView="Form2"
        iViewManager="WindowsFormViewManager"
        name="MyNavigationGraph"
        state="State"
        statePersist="SqlServerPersistState" >
        <node view="Form2">
                <navigateTo navigateValue=" MoreInfo" view="Form3"/>
                <navigateTo navigateValue="DataAdded" view="Form4" />
                <navigateTo navigateValue="fail" view="error" />
        </node>
        <node view="Form3">
                <navigateTo navigateValue="DataAdded" view="Form4" />
                <navigateTo navigateValue="fail" view="error" />
        </node>
                <node view="Form4">
                <navigateTo navigateValue="Restart" view="Form1" />
                <navigateTo navigateValue="fail" view="error" />
        </node>
        <node view="Error">
                <navigateTo navigateValue="Form1" view="Form1" />
                <navigateTo navigateValue="Form2" view="Form2" />
                <navigateTo navigateValue="Form3" view="Form3" />
                <navigateTo navigateValue="Form4" view="Form4" />
        </node>
</navigationGraph>
```

Contoh kode *<navigationGraph>* aplikasi *web*

```
<navigationGraph
        startView="Page2"
        iViewManager="WebFormViewManager"
        name="MyNavigationGraph"
        state="State"
        statePersist="SqlServerPersistState"
        cacheExpirationMode="Sliding"
        cacheExpirationInterval="1000">
        <node view="Page2">
                <navigateTo navigateValue="MoreInfo" view="Page3"/>
                <navigateTo navigateValue="DataAdded" view="Page4" />
                <navigateTo navigateValue="fail" view="error" />
        </node>
        <node view="Page3">
                <navigateTo navigateValue="DataAdded" view="Page4" />
                <navigateTo navigateValue="fail" view="error" />
        </node>
        <node view="Page4">
                <navigateTo navigateValue="Restart" view="Page1" />
                <navigateTo navigateValue="fail" view="error" />
        </node>
        <node view="Error">
                <navigateTo navigateValue="Page1" view="Page1" />
                <navigateTo navigateValue="Page2" view="Page2" />
                <navigateTo navigateValue="Page3" view="Page3" />
                <navigateTo navigateValue="Page4" view="Page4" />
        </node>
</navigationGraph>
```

5) *uipWizard (optional windows)* : menentukan detail dari *graph navigator*. Digunakan untuk yang tidak menggunakan percabangan alur *view*.

Bagian ini adalah cara yang mudah untuk membuat *graph navigator* untuk navigasi *linear*. *Next, Previous, Finish* dan *Cancel* disediakan untuk semua *node*.

Contoh kode *<uipWizard>*

```
<uipWizard name="CarWizard">
        <sequence view="ClientInfo"/>
        <sequence view="CarInfo"/>
        <sequence view="Confirmation"/>
</uipWizard>
```

6) *userControls* (*optional windows*) : menentukan detail untuk *user control navigator*. Dibutuhkan oleh tiap *task* yang menggunakan *user control navigator*. navigationGraph mempunyai attribut :

Tabel II.5 Tabel atribut userControls

startForm	nama dari *view* pertama yang digunakan dalam proses. Ini tidak harus *view* pertama yang ditampilkan kepada pengguna.
iViewManager	nama *view manager* yang di deklarasikan di *<objectTypes>*.
name	nama dari *graph navigator*.
state	nama dari *state* yang di deklarasikan di *<objectTypes>*.
statePersist	nama dari *state persistence* yang di deklarasikan.
cacheExpirationMode (optional)	metode *expire* yang digunakan untuk menidak valid kan *cache*.
cacheExpirationInterval (optional)	waktu yang dibutuhkan untuk *expire*.
name	nama *instance view* dari *form*.
viewName	nama dari *view* dalam *file configuration*.

Contoh kode *<userControls>*

```
<userControls name="Shop"
        startForm="StoreForm"
        iViewManager="WindowsFormViewManager"
        state="State"
        statePersist="SqlServerPersistState">
```

```
        <form name="StoreForm">
                <childView name="shoppingCart" viewName="cart"/>
                <childView name="catalog" viewName="browsecatalog"/>
        </form>
</userControls>
```

II.1.3.2. Controller

Setiap aplikasi yang menggunakan *UIP Application Block* menggunakan 1 atau lebih *controller*, yang secara keseluruhan digunakan untuk mengatur mekanisme antara *user* dengan aplikasi yang ada. *Contoller* terdiri dari metode navigasi yang akan dipanggil oleh komponen *user interface* untuk menentukan *view* apa yang akan ditampilkan kepada pengguna, metode fungsional yang dipanggil oleh *component user interface*, metode penyimpanan *state* dalam media penyimpanan. *Controller* yang digunakan dalam aplikasi ditentukan dalam *configuration file.*

Controller yang digunakan merupakan turunan dari *ControllerBase* yang telah mempunyai metode untuk:

1) Membaca dan menulis state dari task yang sedang berlangsung.
2) Mengontrol navigasi dengan mengatur navigate values.
3) Mendapatkan data dari task yang dijalankan atau dilanjutkan.

II.1.3.3. Controller Base

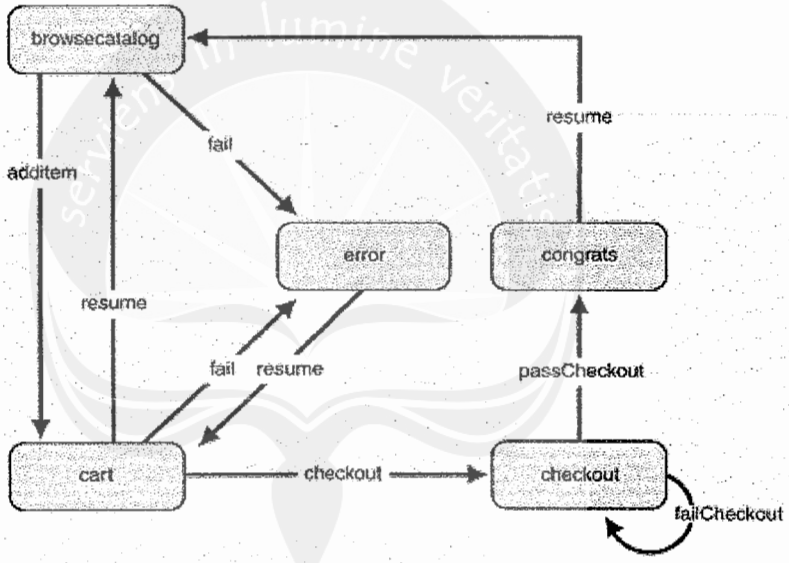
ControllerBase adalah *class controller standar* yang ada dalam *UIP Appplication Block* dimana kita dapat gunakan untuk membuat *class control* spesifik aplikasi.

II.1.3.4. Graph Navigator

Graph Navigator adalah tipe *navigator* yang menggunakan untuk menentukan *user interface control flow* dari suatu *view* ke *view* yang lain.

II.1.3.5. Navigation Graph

Navigation Graph terdiri dari *node* awal, *node* akhir, dan 0 atau lebih *node* diantaranya. Jalan antara *node* awal dan *node* akhir bisa saja hanya simple *linear*, atau juga bisa menggunakan percabangan ataupun *looping*. Ini dapat dilihat dari contoh dibawah:



Gambar II.3 Contoh Navigation Graph

Kode dari *Navigation Graph* tersebut:

```
<navigationGraph
        iViewManager="WinFormViewManager"
        name="Shopping"
        state="State"
        statePersist="SqlServerPersistState"
        startView="cart">
        <node view="cart">
                <navigateTo navigateValue="resume" view="browsecatalog" />
                <navigateTo navigateValue="checkout" view="checkout" />
                <navigateTo navigateValue="fail" view="error" />
```

```
        </node>
        <node view="browsecatalog">
                <navigateTo navigateValue="addItem" view="cart"/>
                <navigateTo navigateValue="fail" view="error" />
        </node>
        <node view="error">
                <navigateTo navigateValue="resume" view="cart" />
        </node>
        <node view="checkout">
                <navigateTo navigateValue="passCheckout" view="congrats" />
                <navigateTo navigateValue="failCheckout" view="checkout" />
        </node>
        <node view="congrats">
                <navigateTo navigateValue="resume" view="browsecatalog" />
        </node>
</navigationGraph>
```

II.1.3.6. Navigator

Navigator menentukan interaksi antara *view* dan *controller* dengan membuat *view manager* dan menggunakannya untuk menjalankan kode khusus menurut tipe application. *UIP* menyediakan 4 tipe navigator yaitu: *graph navigator, open navigator, user control navigator,* dan *wizard navigator.*

II.1.3.7. Open Navigator

Open navigator adalah tipe *navigator* yang tidak mengharuskan kita menentukan transisi yang diperbolehkan dalam *configuration file.* Transisi yang ada menuju pada kode dalam aplikasi.

II.1.3.8. Shared Transitions

Shared Transition adalah transisi umum untuk banyak atau semua *node* dalam sebuah *graph navigator* atau antar *graph navigator.*

II.1.3.9. State Persistence Provider

State Persistence Provider adalah *component* yang mengimplementasikan *IStateProvider* yang membungkus mengeluarkan dan menyimpan state dari proses dari suatu posisi state tertentu. Sebuah *state persistence provider* diasosiasikan pada setiap *user interface proses* dalam *file configuration*. *UIP Appplocation Block* menyediakan beberapa *state persistence provider* :

1) *IsolatedStoragePersistence*
   Untuk proses yang dibutuhkan dalam beberapa *interaction* ataupun *user* dan disimpan dalam *isolated storage*.
2) *SecureIsolatedStoragePersistence*
   Untuk proses yang dibutuhkan dalam beberapa *interaction* ataupun *user* dan disimpan dalam *isolated storage* serta mengandung informasi sensitif (biasanya di-*encript*).
3) *SQLServerPersistState*
   Untuk proses yang dibutuhkan dalam beberapa *interaction* ataupun *user* pada *database*.
4) *SecureSQLServerPersistState*
   Untuk proses yang dibutuhkan dalam beberapa *interaction* ataupun *user* pada *database* serta mengandung informasi sensitif (biasanya di-*encript*).
5) *SessionStatePersistence*
   Untuk proses dalam *web* aplikasi yang mempunyai jangka waktu sama bahkan lebih pendek dari *session* untuk *single user*.
6) *MemoryStatePersistence*

Untuk proses dalam *windows* aplikasi yang mempunyai jangka waktu selama aplikasi berjalan dan hanya cocok untuk *single user*.

Untuk tingkat yang lebih lanjut kita juga dapat membuat dan memodifikasi sendiri sebuah *state persistence provider* dengan membuat kelas yang mengimplementasikan *IStateProvider* dan menentukan sendiri algoritma untuk penyimpanan dan pengaksesan *state*.

II.1.3.10. State

State menggambarkan *model* dari pola *MVC*. Kita dapat menggunakan *state* untuk menyimpan status *task* yang sedang dilalui proses *user*. *State* juga mempunyai metode yang memungkinkan kita untuk mengambil informasi khusus dalam *state information*, menyimpan *state* dalam lokasi lain, memberitahukan *view* bahwa *state* telah berubah. *Controller* dan *view manager* harus mempunyai cara untuk mengetahui dimana *user* dalam proses dan kemana *user* akan pergi berikutnya, sebagai contoh, dalam aplikasi *shopping chart*, jika *user* menunda belanja dan suatu saat kembali, maka aplikasi harus mempunyai cara untuk mendapatkan lokasi kembali *user* tersebut dan informasi lain yang dibutuhkan.

Class State terdiri dari beberapa properties:

1) *CurrentView* : mengembalikan *view* saat ini dari *navigator*.
2) *NavigateValue* : mengembalikan *view* berikutnya dari *navigator*.
3) *Navigator* : mengembalikan *navigator*.
4) *TaskId* : mengembalikan *task id* saat ini.

5) *SyncRoot* : digunakan untuk akses sinkronisasi ke *hash table internal.*

Class state juga mempunyai beberapa metode:

1) *Add* : menambahkan informasi ke *instance* yang ada.
2) *Remove* : membuang informasi dari *instance* yang ada.
3) *Contains* : mengecek apakah item tertentu ada.
4) *Accept* : menyimpan dari *state persistence provider* ke *State.*
5) *Save* : menyimpan *State* ke *state persistence provider.*
6) *CopyTo* : meng-*copy* isi *State* kedalam *array.*

II.1.3.11. State Type

Semua *controller* dan *views* yang berinteraksi dalam sebuah proses berbagi *state* proses. *State* ini dapat dibagi menjadi beberapa tipe sehingga *controller* dan *views* mendapat keuntungan dari pengecekan saat *design. UIP Application Block* memiliki *state* yang bertindak sebagai kamus, tapi kita depat mendefinisikan sendiri tipe *state* kita untuk sebuah proses jika kita membutuhkan keuntungan tersebut. *Object state* dapat memicu *event* untuk sebuah *view* dengan memberitahukan perubahan mereka sehingga mereka dapat berubah berdasarkan itu.

II.1.3.12. Task

Task adalah proses yang berjalan dari sebuah *user interface proses.* Sebagai contoh, seorang *user* dapat memulai *task user interface proses* "Membuat pelanggan baru". *Task* yang membungkus semua perubahan *state*

antara aplikasi yang user disebut dengan *TaskId*. Seorang *user* dapat menjalankan beberapa *task* yang berbeda bersamaan.

II.1.3.13. Task State

Task State adalah *state* sekarang dari sebuah proses. *State* ini dibagi menjadi 2 bagian yaitu: *state* yang digunakan aplikasi dan *state* yang merepresentasikan proses tersebut dalam *user interface proses*. Sebuah *controller* dapat mengakses *task state* dengan menggunakan *property State* dalam *base class*. *State* disimpan dalam *State Persistence Provider*.

II.1.3.14. Task State Location

Task state location menentukan dimana *task state* dari proses *user interface* yang sedang berjalan disimpan. Ini bisa di semua penyimpanan informasi yang *valid*, tapi ketika memutuskan dimana letak nya dalam sebuah aplikasi harus diperhatikan :

1) *Target Platform* dari *user interface*.
2) Apakah *state* harus dikembalikan diantara *session*.
3) Apakah *state* harus dikembalikan ketika aplikasi diulang/*restart*.
4) Kebutuhan keamanan.

Untuk mengakses dan menyimpan *state* dari lokasi yang berbeda diimplementasikan dalam *state persistence provider*.

II.1.3.15. Task Id

Task Id menentukan proses apa yang sedang berjalan dalam aplikasi. Aplikasi dapat menggunakan *task id* untuk melanjutkan sebuah proses ataupun mengasosiasikan sebuah *task* dengan seorang *user*.

II.1.3.16. UIP Manager

Class UIP Manager digunakan sebagai jalan masuk ke *User Interface Process*. Jika kita menggunakan *UIP Application Block*, aplikasi kita harus mengandung kode yang memanggil *UIP Manager* untuk menjalankan *task* dan menjalankan *navigator* yang tepat.

II.1.3.17. User

User adalah siapa saja yang berinteraksi dengan aplikasi yang menjalankan *task*. Jika *task* membutuhkan interaksi dari 1 atau lebih *user* untuk berhasil, maka aplikasi harus diasosiasikan dengan dengan *user* yang tepat pada saat yang tepat.

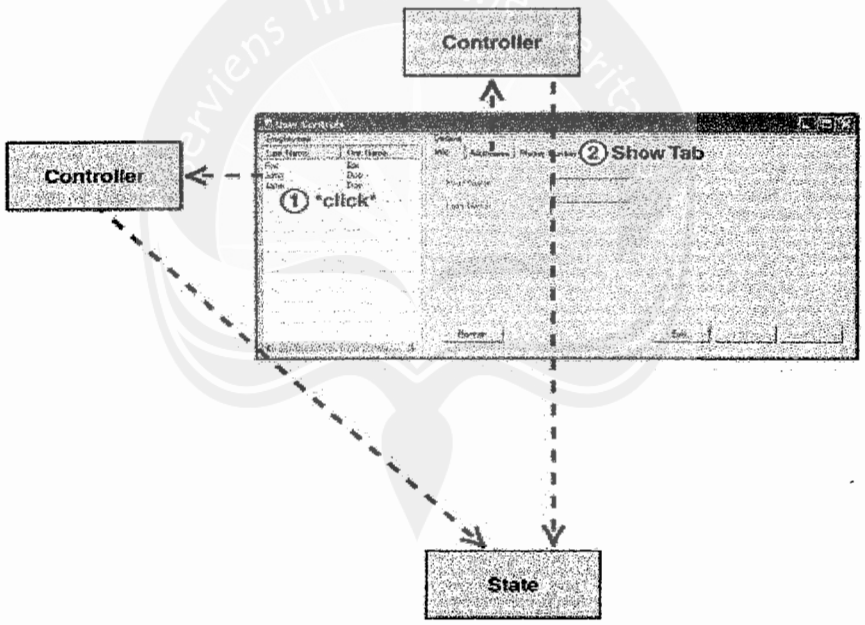
II.1.3.18. User Control Navigator

Aplikasi berbasis *windows* sering mempunyai *user control* yang muncul sebagai bagian dari sebuah *form* atau dari *control* lain. *User control navigator* memberikan mekanisme navigasi yang berfokus dari satu *control* dan memberikan fleksibilitas untuk bertukar *control*.

UIP Application Block menggunakan class *UserControlsNavigator* untuk menyediakan mekanisme navigasi yang berfokus pada 1 *control* pada lainnya dalam 1 *form*, selain transisi antar *form*. *Class* ini

memampukan kita mengumpulkan *form* dengan *user control* mereka dengan *model* dan *control* mereka sendiri, dan memampukan kita untuk bertukar *control* dengan menggunakan nama.

Untuk lebih jelas dapat kita lihat contoh dibawah, dimana dalam *form* terdapat *list view* dan *tab control* dengan beberapa *tab*. Dalam tiap *tab* terdiri dari beberapa *control*. Ketika *user* meng-*click* pada *list view* maka akan memicu salah satu dari *tab* menjadi aktif. *List item* dan *tab control* mempunyai *control* yang berbeda, namun mereka menggunakan *object State* yang sama.



Gambar II.4 Contoh User Control Navigator

II.1.3.19. User Interface Process

User Interface Process adalah kumpulan kegiatan *user* dalam mencapai tujuan. Sebagai contoh : melihat katalog, *cheking out*, mendaftarkan *credit card* baru adalah contoh *user interface process* untuk aplikasi penjualan.

User interface process terdiri dari:

1) *Name*.
2) *Navigator*.
3) *State Location*.
4) *State Type*.

II.1.3.20. View

View adalah komponen *user interface* yang sebenarnya dari aplikasi kita. Untuk aplikasi *windows*, kita menggunakan *windows forms*. Sedangkan untuk aplikasi *web* kita menggunakan *web pages*. *UIP Application Block* menyediakan sebuah *interface* (*IView*) yang harus diimplementasikan oleh *views* yang ada. Disini juga sudah tersedia *class* yang mengimplementasikan *IView* yaitu *WindowsFormView*, *WindowsFormControlView*, dan *WebFormView*.

II.1.3.21. View Activation

View manager mengaktifkan *view* dan melemparkan *control* ke dalamnya. Secara spesifik bagaimana sebuah view diaktifkan tergantung pada tipe aplikasinya. Sebagai contoh, aplikasi *web* menggunakan metode *Response.Redirect*(), sedangkan untuk aplikasi *windows* menggunakan metode *Show*(). Beberapa tipe aplikasi memungkinkan mengaktifkan 1 atau lebih *view* dalam saat yang bersamaan.

II.1.3.22. View Manager

View manager adalah komponen yang membungkus semua pekerjaan yang berkaitan dengan mengaktifkan *view* dan mengasosiasikan sebuah *view* dengan *task* tertentu. *View*

manager mengimplementasikan *IViewManager* dalam *UIP Application Block*. Jadi kita juga dimungkinkan untuk membuat *view manager* sendiri dengan membuat *class* yang mengimplementasikan *interface* tersebut.

View manager bertanggung jawab membuat dan mengaktifkan *view* yang diminta oleh *navigator*. Dalam kasus tertentu, kita membutuhkan untuk membuat aplikasi yang mempunyai lebih dari 1 *user interface*. Biasanya kita harus membuat aplikasi *windows* dan *web* untuk aplikasi yang sama. Pengaturan dari *view* tergantung pada tipe aplikasi yang kita buat, sebagai contoh kode untuk menampilkan *windows form* berbeda dengan kode yang digunakan untuk menampilkan *web form*. Karena itu, *manager* yang berbeda bertanggung jawab untuk tipe aplikasi yang berbeda.

Setiap *view manager* dapat dibuat dengan mengimplementasikan *IViewManager*, ada 3 *manager* yang tersedia dalam *UIP Application Block* :

1) *WebFormViewManager* untuk mengatur *view* aplikasi *web*.

   Dalam *class* ini ada beberapa metode yaitu:

   a. *ActivateView* : mengaktifkan *view* yang menjadi paremeter.
   b. *GetCurrentTasks* : mengembalikan *array task GUID* yang sedang dijalankan.
   c. *IRequestCurrentView* : mengecek apakah *view* sekarang adalah *view* yang diminta.
   d. *GetViewNameForCurrentRequest* : mengecek *url* yang diminta dan mendapatkan *view name* dari *url* tersebut.

2) *WindowsFormViewManager* untuk mengatur *view* aplikasi *windows*.

Dalam *class* ini ada beberapa metode yaitu:

a. *ActivateView* : mengaktifkan *view* yang menjadi paremeter.

b. *GetCurrentTasks* : mengembalikan *array task GUID* yang sedang dijalankan.

c. *StoreProperty* : menyimpan *property* dari *form* ke dalam *hash table internal*.

d. *GetActiveViewCount* : mendapatkan *view* yang aktif dalam *manager*.

3) *WizardFormViewManager* untuk mengatur *wizard*.

Dalam *class* ini ada beberapa metode yaitu:

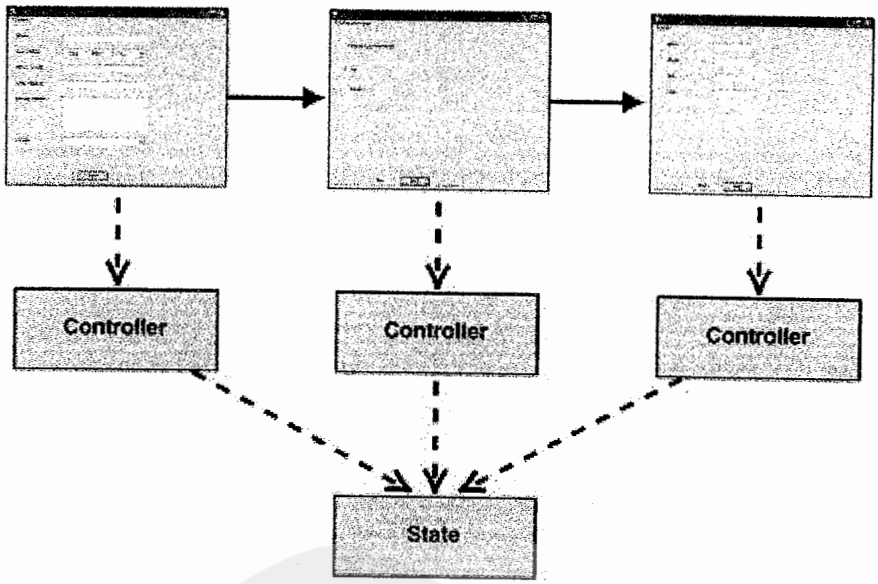
a. *ActivateView* : mengaktifkan *view* yang menjadi paremeter.

b. *GetCurrentTasks* : mengembalikan *array task GUID* yang sedang dijalankan.

II.1.3.23. Wizard Navigator

Wizard Navigator adalah *navigator* yang menyediakan mekanisme navigasi antar *form* sebagai *wizard*. *Wizard Navigator* menyediakan cara untuk mendefinisikan transisi antar *view* dengan menggunakan *cancel*, *back*, *next* dan *finish*.

Wizard lebih umum digunakan untuk aplikasi berbasis *windows*, dan navigasi antar *form* dapat diprediksi. Berikut adalah contoh hubungan antara *view*, *controller* dan *state* dalam *wizard navigator*.



Gambar II.5 Contoh Wizard Navigator