Moving from UI to a Presentation Layer

BUSINESS MAKING PROGRESS™

Peter Judge
pjudge@progress.com

PROGRESS software

UI vs. Presentation Layer

> *User Interface* is the implementation of the intended User Experience in terms of page layout, page transitions and page control elements [1]

Presentation layer is the design and implementation of the code and other resources that provide that UI

[1] peterchen on Stackoverflow.com: “Difference between presentation layer and user-interface” http://stackoverflow.com/questions/2907604/difference-between-presentation-layer-and-user-interface/2925053#2925053

The Beginning: A Very Good Place To Start

BUSINESS MAKING PROGRESS

```
for each Customer:
  display CustNum
    with frame f1Cust 1 down.
end.
```

PROGRESS
software

BUSINESS MAKING PROGRESS

The Beginning: A Very Good Place To Start

```
for each customer:
  displ custnum
       with frame fCust 1 down.
end.
```

What do we have here? A way to

… get data

… show data

… process inputs

The Beginning: A Very Good Place To Start

```
for each customer:
  displ custnum
       with frame fCust 1 down.

end.
```

What do we have here? A way to

… get data

… show data

… process inputs

The Beginning: A Very Good Place To Start

```
for each customer:
  displ custnum
       with frame fCust 1 down.

end.
```

What do we have here? A way to

… get data

… show data

… process inputs

The Beginning: A Very Good Place To Start

```
for each customer:
   displ custnum
        with frame fCust 1 down.

end.
```

What do we have here? A way to

… get data

… show data

… process inputs

An Important Improvement

```
define temp-table ttCustomer no-undo
     like Customer.

run GetCustomers
   (input cWhereClause,
    output temp-table ttCustomer).

for each ttCustomer:
  displ custnum
    with frame fCust 1 down.

end.
```

A Further Modest Improvement


```
define temp-table ttCustomer no-undo
      field CustNum as integer
      /* fields */.
hAppServer
      :connect("-AppService asPugChallenge11").

run GetCustomers on hAppServer
       (input cWhereClause,
        output temp-table ttCustomer).

for each ttCustomer:
  displ custnum
    with frame fCust 1 down.

end.
```

Time to look at real* code

* relatively real-ish, at any rate

Recap Part 1

```
customerwin.w
   GetCustomerData on hAppServer
   define query on ttCustomer
   query navigation
   define UI (frames)
   define buttons &     ‘on choose’ triggers
   perform ui logic
   hold things together
```

Recap Part 2

BUSINESS MAKING PROGRESS.

```
customer_eventhandler.p
   perform ui logic
   hold things together

customer_clientdata.p
   getCustomerData on hAppServer
   define query on ttCustomer
   query navigation etc

customer_abl_gui.p
customer_net_gui.cls
customer_web_ui.p
   define UI (frames, controls, pages)
   define buttons / links & ‘on choose’ triggers
```

Tying the pieces together

Presentation Layer Design

BUSINESS MAKING PROGRESS.

Always working with 3 parts

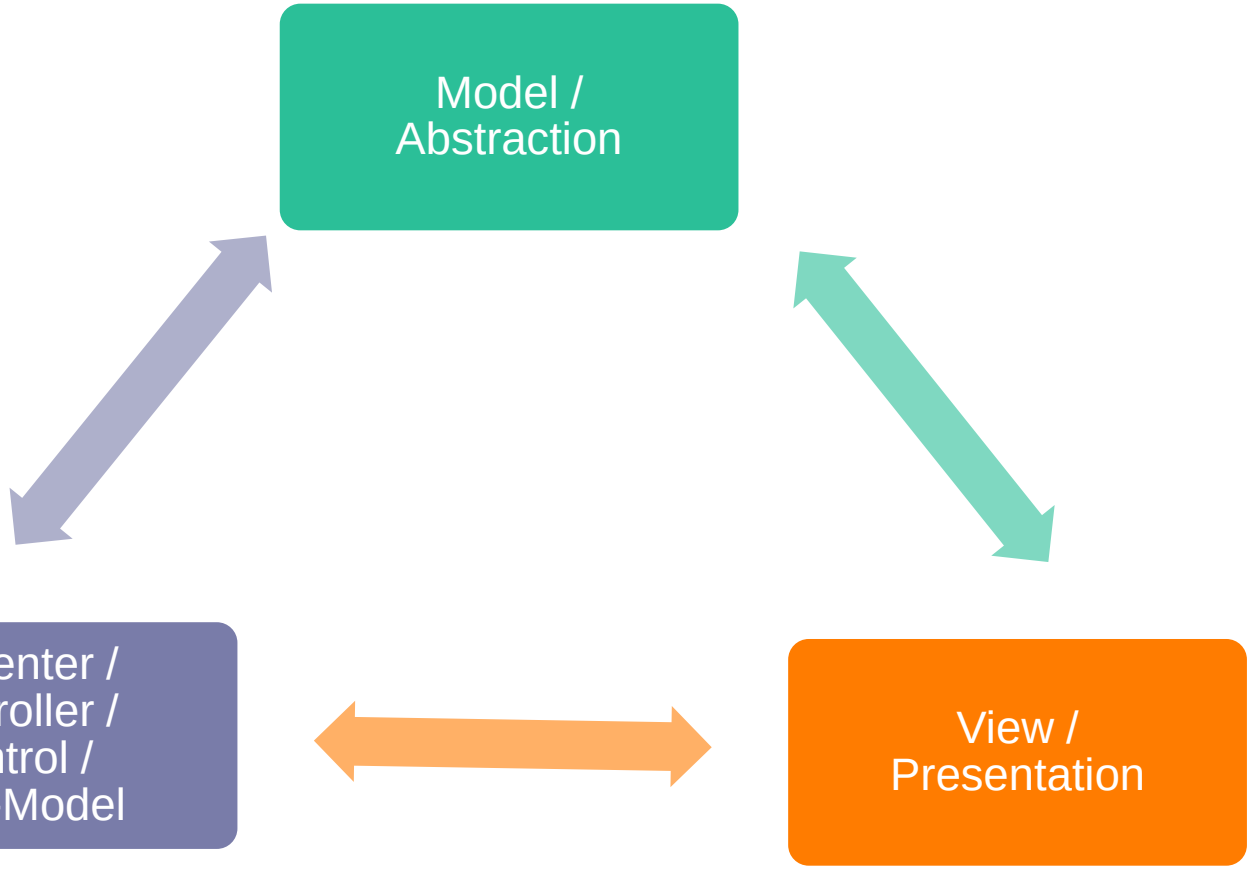
1. Something to hold data
2. Something to show the data to a user
3. Something to receive the user's actions

We also need something to hold everything together

- Multiple designs that combine these 3 concepts
  - MVC (model-view-controller)
  - PAC (presentation-abstraction-control)
  - MVP (model-view-presenter)
  - MVVM (model-view-view-model)

Presentation Layer Design

Contains local data, access to business logic



Handles inter-component communication; Can contain UI logic

Draws UI; Receives user input

Model-View-Controller (MVC)

BUSINESS MAKING PROGRESS

- MVC Granddaddy of them all
  - All communication through Controller

Mouse Click ⇨ *Controller* ⇨ *Model* ⇨ *Controller* ⇨ *View*

- PAC (Presentation-Abstraction-Control)
  - Used as hierarchical structure of agents
  - Triad of presentation, abstraction and control parts
  - Triads only communicate through control part

- MVP (Model-View-Presenter)
  - Derivative of MVC, mostly for building UI
  - Presenter becomes "middle-man"
  - View is responsible for handling UI events

Mouse Click ⇨ *View* ⇨ *Presenter* ⇨ *Model* ⇨ *View*
⇨ *Presenter* ⇨ *View*

Model-View-ViewModel (MVVM)

BUSINESS MAKING PROGRESS.

Derivative of PresentationModel :

> The essence of a *Presentation Model* is of a fully self-contained class that represents all the data and behavior of the UI window, but without any of the controls used to render that UI on the screen[1]

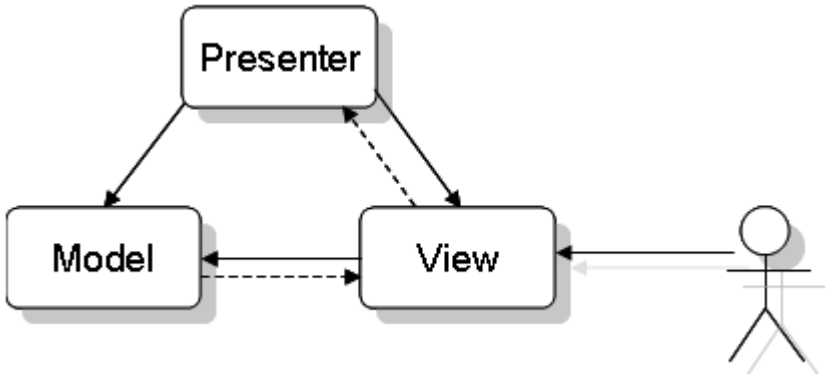
> MVVM a specialization of this the more general PM pattern, tailor-made for the WPF and Silverlight platforms[2]

Model and View are same as for MVP/MVC

1. Martin Fowler, **Presentation Model (** http://martinfowler.com/eaaDev/PresentationModel.html)
2. Josh Smith, **WPF Apps With The Model-View-ViewModel Design Pattern** http://msdn.microsoft.com/en-us/magazine/dd419663.aspx

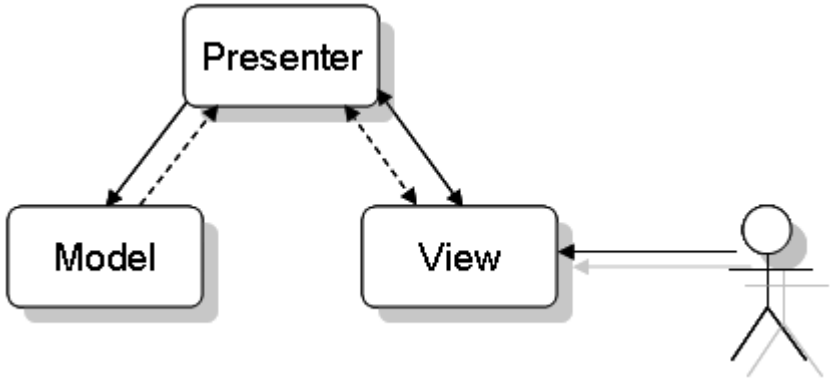
What does OERA use?

MVP Pattern

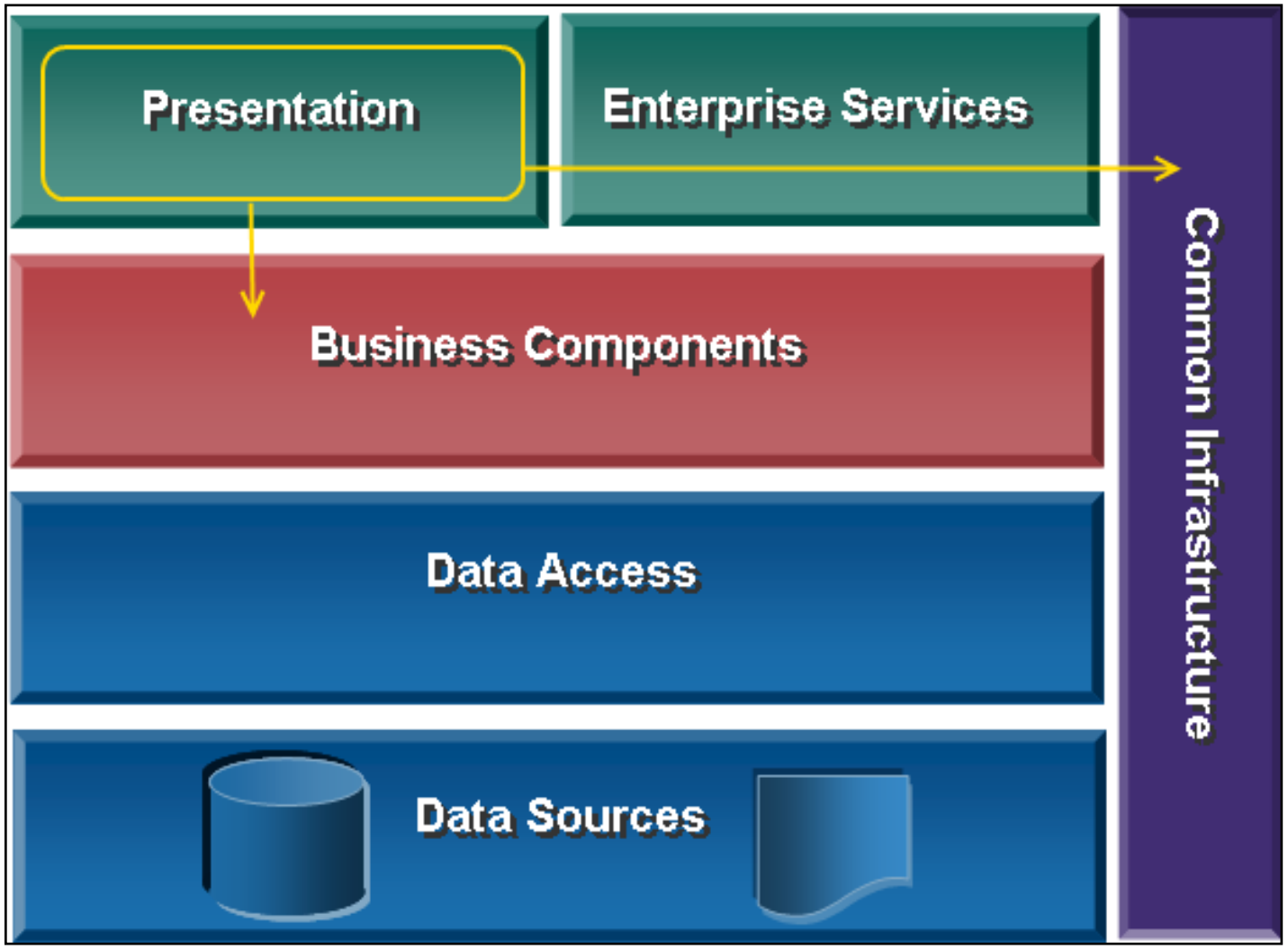


- Supervising Presenter
  - Good fit for ABL tight data binding
  - Presenter does **some** Work

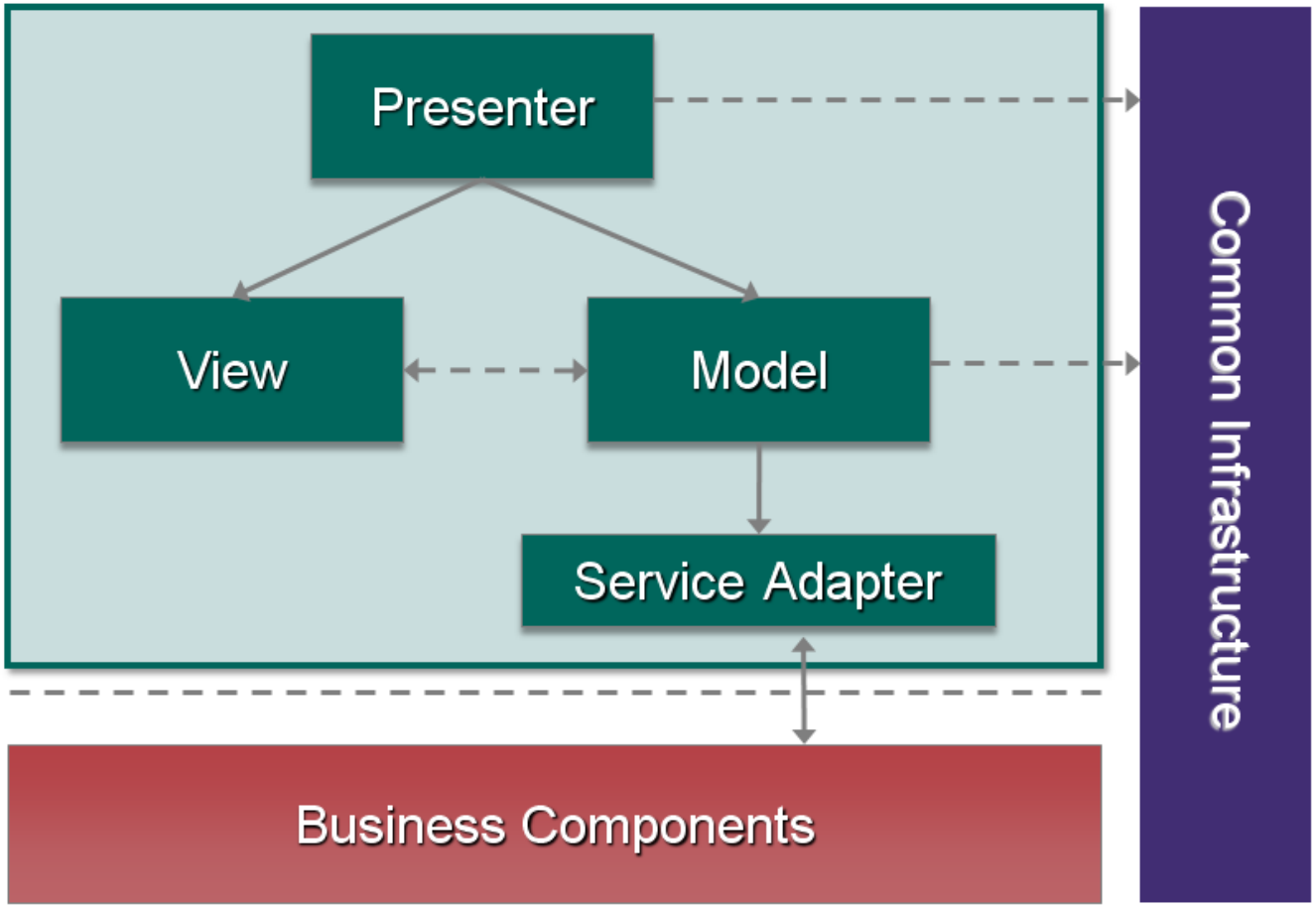
- Passive View
  - View, Model have no dependencies
  - Presenter does **all** Work



OpenEdge Reference Architecture



MVP in the OERA Presentation Layer



Tying it back to the code

BUSINESS MAKING PROGRESS

customer_presenter.p customer_eventhandler.p

perform ui logic
hold things together

customer_model.p customer_clientdata.p

getCustomerData on hAppServer
define query on ttCustomer
query navigation etc

customer_view_window.p customer_abl_gui.p
customer_view_form.p customer_net_gui.cls
customer_view_page.p customer_web_ui.p

define UI (frames, controls, pages)
define buttons / links & ‘on choose’ triggers

Tying it back to the code

BUSINESS MAKING PROGRESS

customer_presenter.p customer_eventhandler.p

perform ui logic
hold things together

customer_sdo_model.p
customer_json_model.p
customer_db_model.p
customer_be_model.p customer_clientdata.p

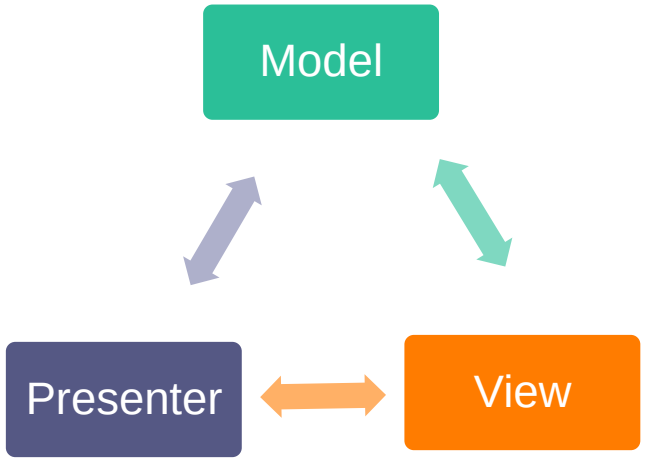
getCustomerData on hAppServer
define query on ttCustomer
query navigation etc

customer_view_window.p customer_abl_gui.p

define UI (frames, controls, pages)
define buttons / links & ‘on choose’ triggers

Summary

- Reusable, consistent logic across UI technologies
  - We already knew about business logic reuse
  - Now have reusable UI logic
- Better separation of concerns
  - View does UI
  - Model does Data
  - Presenter does Thinking
- Supports incremental migration
- Easier testing, clearer points of failure



Further Reading

- OERI Model View Presenter Architecture
  http://communities.progress.com/pcom/docs/DOC-105108
  - AutoEdge|TheFactory
    http://communities.progress.com/pcom/community/psdn/openedge/architecture/autoedgethefactory
  - OERA Presentation Layer
    http://communities.progress.com/pcom/docs/DOC-34978
- MVC http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html
  - Portland Pattern Repository
    MVC http://c2.com/cgi/wiki?ModelViewController
    MVP http://c2.com/cgi/wiki?ModelViewPresenter
- MVVM
  - http://msdn.microsoft.com/en-us/magazine/dd419663.aspx
  - http://www.nikhilk.net/Silverlight-ViewModel-Pattern.aspx
  - http://en.wikipedia.org/wiki/Model_View_ViewModel
  - http://www.nikhilk.net/View-ViewModel-Interaction.aspx
- "Build your own CAB" (nice overview of many topics)
  http://codebetter.com/jeremymiller/2007/07/26/the-build-your-own-cab-series-table-of-contents/

PROGRESS
software

BUSINESS MAKING PROGRESS

Peter Judge pjudge@progress.com

BUSINESS MAKING PROGRESS™

PROGRESS REVOLUTION

Sept. 19 – 22, 2011
Boston Westin Waterfront Hotel
and Boston Convention & Exhibition Center

PROGRESS software