Session : 1017

Title : Future proofing your application through layering

Subtitle : How to support multiple UI platforms simultaneously

Frank Hilhorst
President Progressive Consulting Inc.
Email: Frank@ProgressiveConsultingInc.com
Phone #: 561-8432839

What is the focus of this presentation?

- Examine how we can support multiple UI platforms simultaneously with minimum duplication of effort

Where does the need to support multiple UI's simultaneously arise from?

- You are migrating or extending an application to a new UI platform
- The old application still needs to be supported
- The old application is still being sold
- The need to make functional changes to the old (and new) application continues during the migration
- The old and the new application share (at least some of) the same screens
- You cannot afford duplicate development teams

What you will learn today

- What forms of support PROGRESS offers for non-native UI technologies
- Understanding the Model/View/Presenter design pattern
- How this pattern can be used to achieve UI independence
- Classifying UI technologies in terms of the layers needed
- Use layering as a vehicle to understand the advantages of ABL/OO

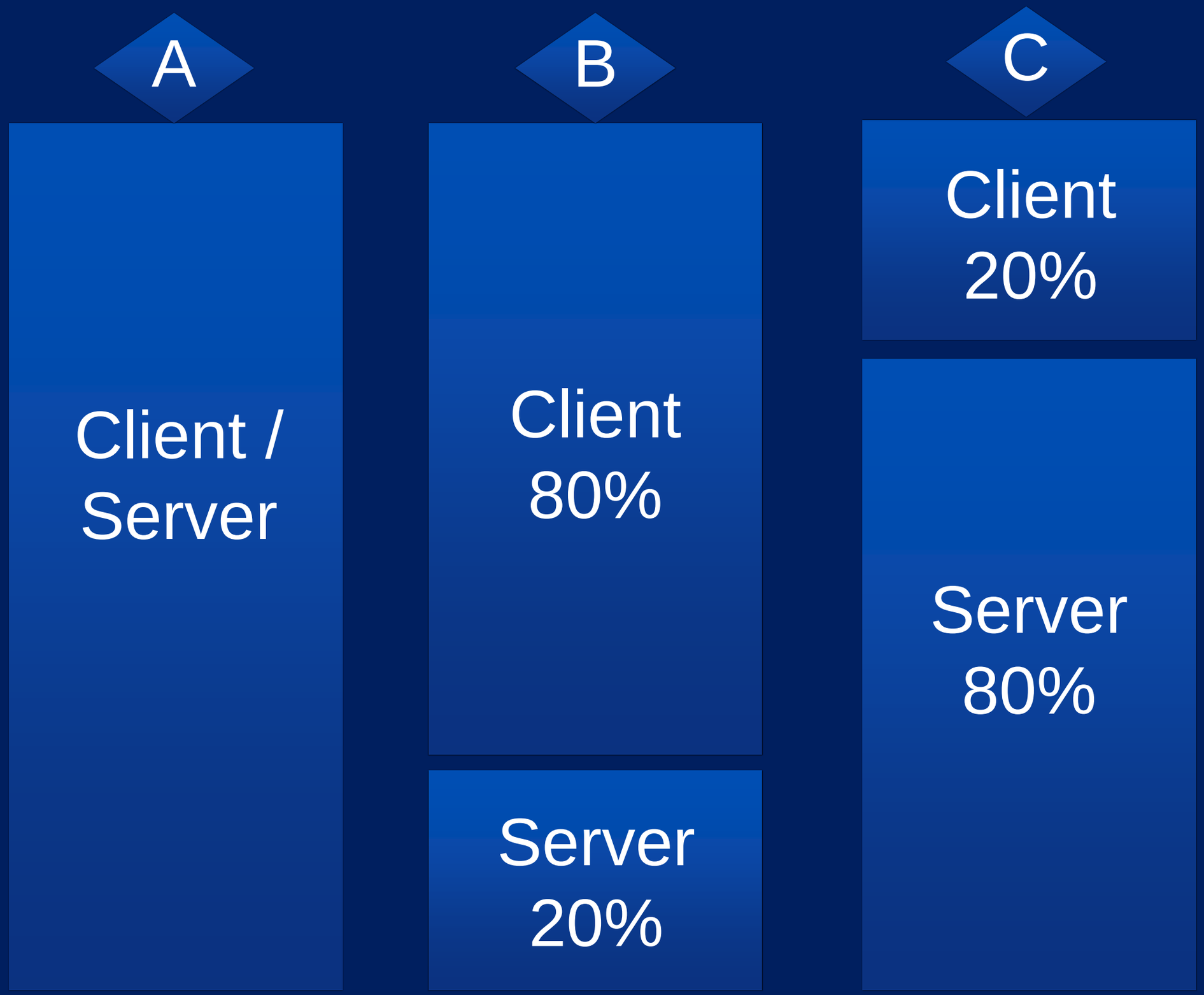
Demo

- Same screen
  - Progress WebClient
    - AppServer implementation
    - Web Services implementation
  - Java Swing
  - OpenLaszlo/Ajax
  - HTML/Javascript/Ajax
- Supported by the same code base

Progress support for non-native UI platforms

- OpenClient (JAVA, .NET etc)
- WebSpeed
- Web Services
- ODBC/JDBC

Which application is easier to port?



The Model/View/Presenter design pattern

Model

Responsibilities:

- Process Input Presenter Layer
- Update State (e.g.Update DB)
- Present data to presenter layer

Presenter

Responsibilities:

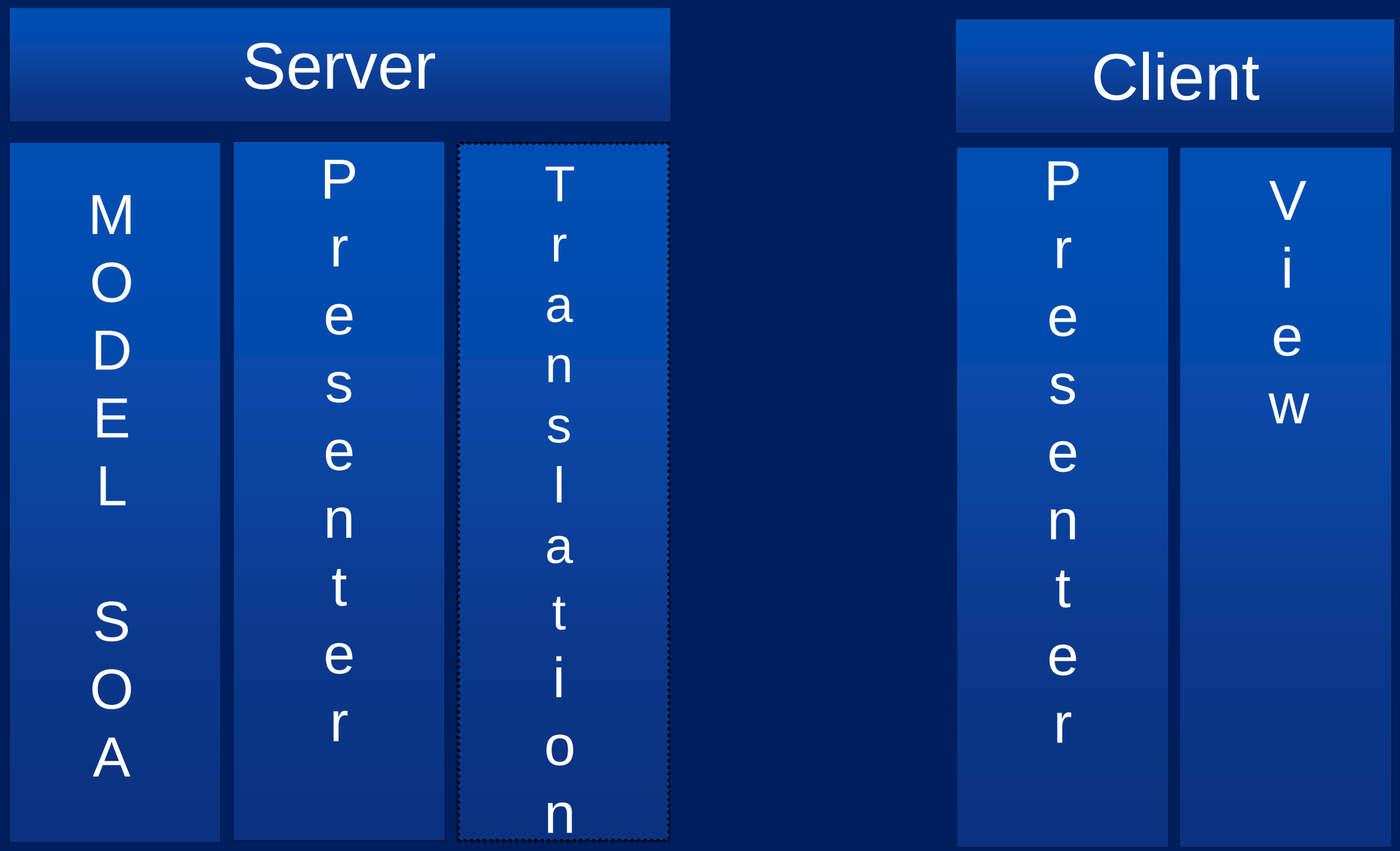
- Process UI Event
- Notify Model and get data
- Present data to view layer

View

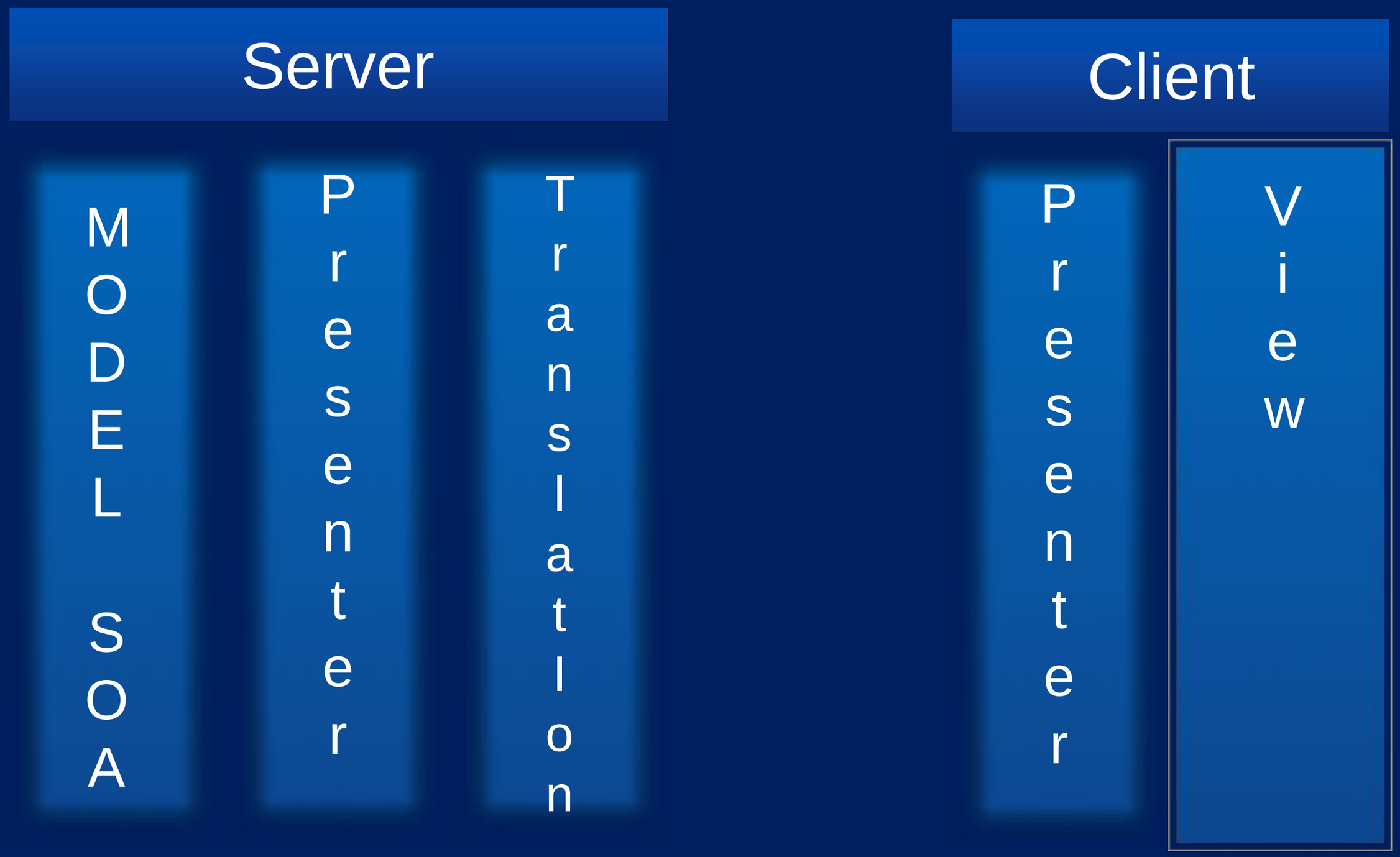
Responsibilities:

- Notify presentation Layer of UI Events
- Update UI state

Super-imposing Model/View/Presenter on the disconnected Model



Deconstructing the view layer



Looking at a UI as a collection of "Event/State Change" relationships

Account Group -> Accounting Info

Scrn Help

Account Group:

Group Code	Name	Description
JT8CC	JT8D COMBUSTION CHAMBER	JT8D COMBUSTION CHAMBER

Transaction Group: Purchasing & A/P

Transaction Type: PO Receipt (RCV)

Transaction Sub Type: Stocked Items

Use: Simple View / Advanced View

G/L Accounts Used:

Debit Entries

G/L Account #	Desciption	Configured in
005120020001100000	Inventory - Finished Goods - Material	Configured in Account Code # I10
005120020001110000	Inventory - Finished Goods - Labor	Configured in Account Code # I11
005120020001120000	Inventory - Finished Goods - Overhead	Configured in Account Code # I12
005120020001130000	Inventory - Finished Goods - Outside Service	Configured in Account Code # I13

Credit Entries

G/L Account #	Desciption	Configured in
005120020001100000	Inventory - Finished Goods - Material	Configured in Account Code # I10
005120020001110000	Inventory - Finished Goods - Labor	Configured in Account Code # I11
005120020001120000	Inventory - Finished Goods - Overhead	Configured in Account Code # I12
005120020001130000	Inventory - Finished Goods - Outside Service	Configured in Account Code # I13

Occurs when:

Ordered items on PO are received into stock.

Notes:

Normally only the material account will be debited.

On Value-Changed of "Transaction Group":
- GetData(Output cList-items)
- TransactionType:List-Items = cListItems.

On Value-Changed "Transaction Type":
- GetData(Output cList-items,Output table ttGlInfo)
- Transaction Sub Type:List-Items = cListItems.
- Populate Debit & Credit info

On Value-Changed "Transaction Sub Type":
- GetData(Output table ttGlInfo)
- Populate Debit & Credit info

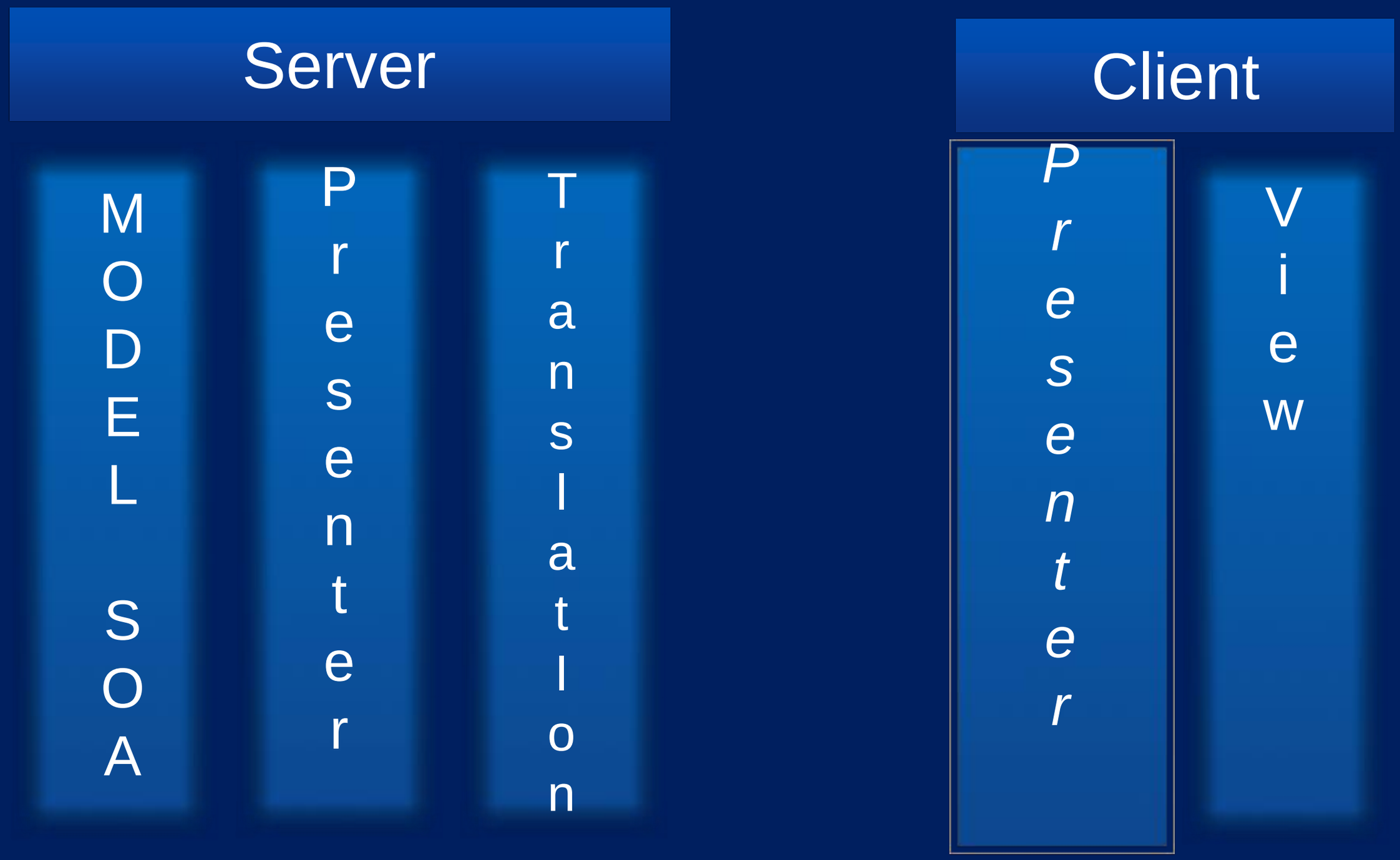
This is all the UI needs to support all "Event/State Change" relationships

Procedure Editor - C:\Progress Presentation\Code Extract\CopdeExtract1.p

File Edit Search Buffer Compile Tools Options Help

```
PROCEDURE GetAccountGroupInfo :
   DEFINE INPUT  PARAMETER icGroupCd AS CHARACTER   NO-UNDO.
   DEFINE OUTPUT PARAMETER ocGroupNm AS CHARACTER   NO-UNDO.
   DEFINE OUTPUT PARAMETER ocGroupDesc AS CHARACTER   NO-UNDO.
   .....
END PROCEDURE.

PROCEDURE GetTransactionGroups :
   DEFINE OUTPUT PARAMETER ocListItemPairs AS CHARACTER   NO-UNDO.
   DEFINE VARIABLE vlOperationSuccceeded AS LOGICAL    NO-UNDO.
   DEFINE VARIABLE vcValMsg AS CHARACTER   NO-UNDO.
   ....
END PROCEDURE.

PROCEDURE ProcessTrType :
   DEFINE INPUT  PARAMETER icGroupCd AS CHARACTER   NO-UNDO.
   DEFINE INPUT  PARAMETER icTransactionType AS CHARACTER   NO-UNDO.
   DEFINE OUTPUT PARAMETER TABLE FOR ttAcctGrpGlAccount.
   DEFINE OUTPUT PARAMETER ocSubTypeListItemPairs AS CHARACTER   NO-UNDO.
   DEFINE OUTPUT PARAMETER ocOccursWhen AS CHARACTER   NO-UNDO.
   DEFINE OUTPUT PARAMETER ocDocNote AS CHARACTER   NO-UNDO.

END PROCEDURE.

PROCEDURE GetTransactionTypes :
   DEFINE INPUT  PARAMETER icTransactionGroupCd AS CHARACTER   NO-UNDO.
   DEFINE INPUT  PARAMETER icViewMode AS CHARACTER   NO-UNDO.
   DEFINE OUTPUT PARAMETER ocListItemPairs AS CHARACTER   NO-UNDO.
   ....
END PROCEDURE.

PROCEDURE ProcessTrSubType :
   DEFINE INPUT  PARAMETER icGroupCd AS CHARACTER   NO-UNDO.
   DEFINE INPUT  PARAMETER icTransactionSubType AS CHARACTER   NO-UNDO.
   DEFINE OUTPUT PARAMETER TABLE FOR ttAcctGrpGlAccount.
   DEFINE OUTPUT PARAMETER ocOccursWhen AS CHARACTER   NO-UNDO.
   DEFINE OUTPUT PARAMETER ocDocNote AS CHARACTER   NO-UNDO.
   ......
END PROCEDURE.
```

And next...
Deconstructing the client side presentation layer



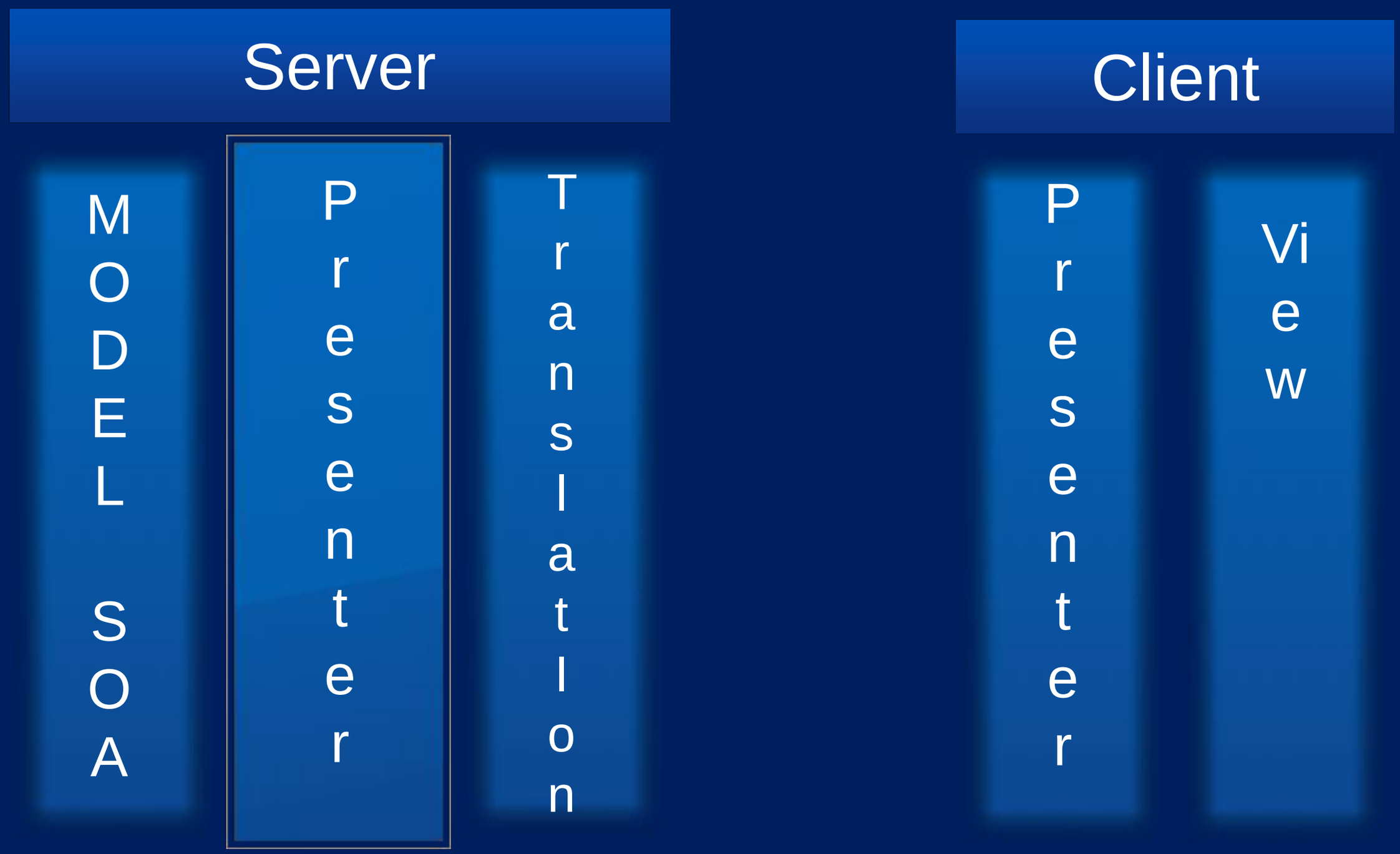
Deconstructing the client side presentation layer

```
METHOD PUBLIC VOID GetTransactionTypes(icTransactionGroupCd AS CHARACTER,
                    icViewMode AS CHARACTER,
                    OUTPUT ocListItemPairs AS CHARACTER ):
  DEFINE VARIABLE vcOutputParameters AS CHARACTER  NO-UNDO.
  DEFINE VARIABLE vlOperationSuccceeded AS LOGICAL   NO-UNDO.
  DEFINE VARIABLE vcValMsg AS CHARACTER  NO-UNDO.

  GetData
    (INPUT "GetTransactionTypes",
     INPUT SUBST("GroupCd=&1`ViewMode=&2",icTransactionGroupCd,icViewMode),
     INPUT-OUTPUT TABLE ttAcctGrpGlAccount,
     OUTPUT ocListItemPairs,
     OUTPUT vlOperationSuccceeded,
     OUTPUT vcValMsg).
  RETURN.
END METHOD.

METHOD PROTECTED VOID GetData(icOperation AS CHARACTER,
                    icInputParameters AS CHARACTER,
                    INPUT-OUTPUT TABLE ttAcctGrpGlAccount,
                    OUTPUT ocOutputParameters AS CHARACTER,
                    OUTPUT olOperationSuccceeded AS LOGICAL,
                    OUTPUT ocValMsg AS CHARACTER ):
    RUN AccountCode/sspAccountGrpDiag.p ON ghAppserver
      (INPUT icOperation,
       INPUT icInputParameters,
       INPUT-OUTPUT TABLE ttAcctGrpGlAccount,
       OUTPUT ocOutputParameters,
       OUTPUT olOperationSuccceeded,
       OUTPUT ocValMsg).
  RETURN.
END METHOD.
```

Each public method uses Protected method “GetData” to communicate with the server side presenter layer.

“sspAccountGrpDiag.p” is the server side “presenter” layer dedicated to comminicating with the UI in the “language” of state changes.

Deconstructing the server side presentation layer



What screen would be easier to create and maintain on another UI Platform?

- A screen that uses 5 APPSERVER procedures
- A screen that uses 1 APPSERVER procedure

A look inside server side presentation layer

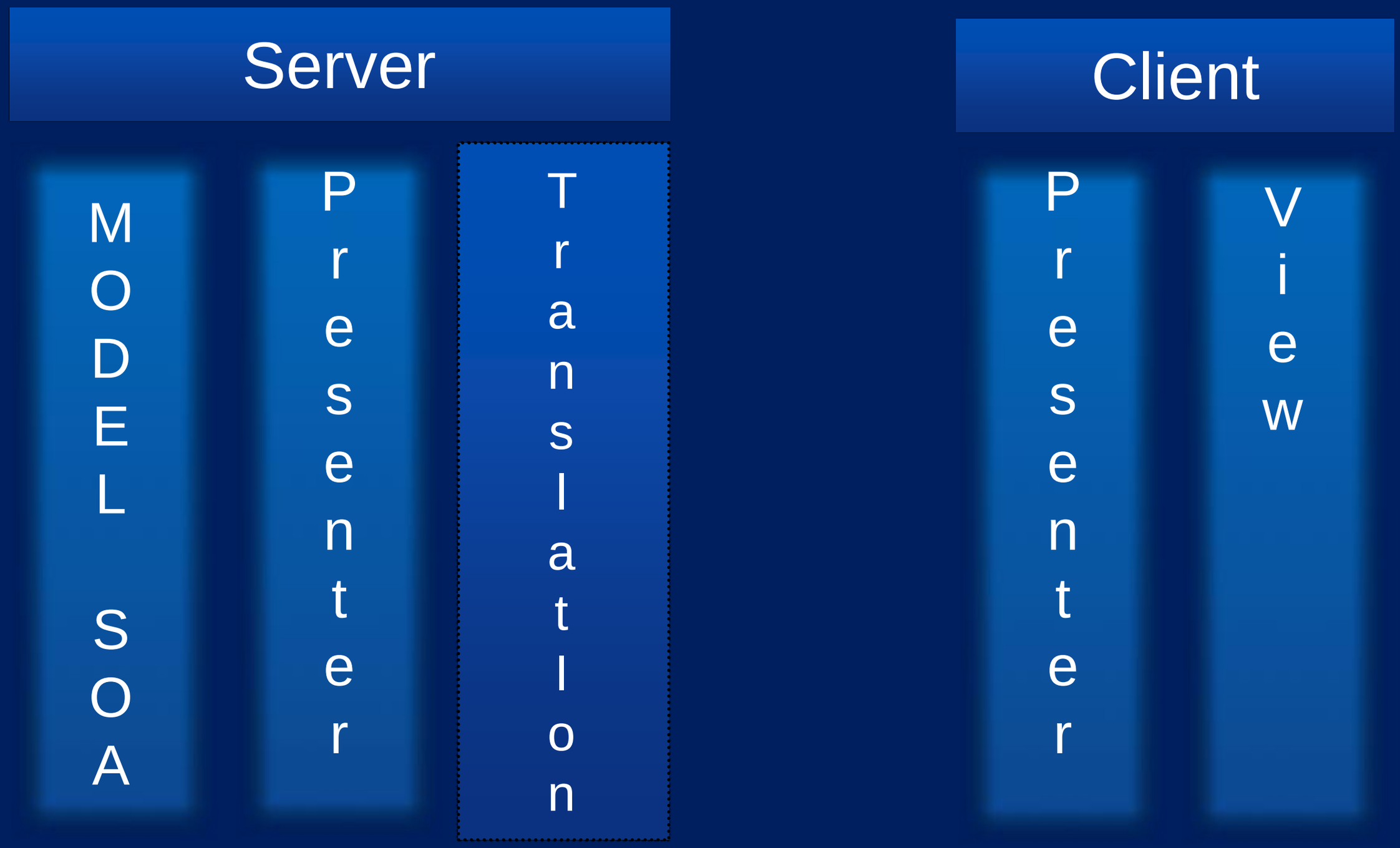
```
/* Genetric input/output parameter signature*/
DEFINE INPUT  PARAMETER icOperation AS CHARACTER   NO-UNDO.
DEFINE INPUT  PARAMETER icInputParameters AS CHARACTER   NO-UNDO.
DEFINE INPUT-OUTPUT PARAMETER TABLE FOR ttAcctGrpGlAccount.
DEFINE OUTPUT PARAMETER ocOutputParameters AS CHARACTER   NO-UNDO.
DEFINE OUTPUT PARAMETER olOperationSucceeded AS LOGICAL    NO-UNDO.
DEFINE OUTPUT PARAMETER ocValMsg AS CHARACTER   NO-UNDO.

/* ***************************  Main Block  *************************** */
RUN VALUE(icOperation).

FUNCTION ParameterValue
 RETURNS CHARACTER
 ( /* parameter-definitions */
  icParName AS CHAR,
  icInputParameters AS CHAR) :
/*------------------------------------------------------------------------------
 Purpose:
  Notes:
------------------------------------------------------------------------------*/
 DEFINE VARIABLE vcParValue AS CHARACTER   NO-UNDO.
 DEFINE VARIABLE vI AS INTEGER    NO-UNDO.
 DEFINE VARIABLE vJ AS INTEGER    NO-UNDO.
 DEFINE VARIABLE vcEntry AS CHARACTER   NO-UNDO.
 DO vI = 1 TO NUM-ENTRIES(icInputParameters,"`"):
   ASSIGN vcEntry = ENTRY(vI,icInputParameters,"`")
        vJ = INDEX(vcEntry,"=").
   IF vJ > 1 AND TRIM(ENTRY(1,vcEntry,"=")) EQ icParName THEN
   DO:
       vcParValue = SUBSTR(vcEntry,vJ + 1).
       LEAVE.
   END.
 END. /*DO vI = 1 TO NUM-ENTRIES(icInputParameters,"`"):*/
 RETURN vcParValue.   /* Function return value. */
END FUNCTION.
```

Generic signature supports all possible inputs and outputs for communicating with the client side “presenter” layer.

icOperation contains the name of the internal procedure to run.

Function “ParameterValue” is used to parse parameter values out of the name value pair string icInputParameters

Examining the Translation Layer



So how do you communicate with a UI technology that only understands XML?

```
icOperation = get-value("Operation").
icInputParameters = REPLACE(get-value("InputParameters"),"***","=").
RUN outputHeader.

RUN AccountCode/sspAccountGrpDiag.p
  (INPUT icOperation,
   INPUT icInputParameters,
   INPUT-OUTPUT TABLE ttAcctGrpGlAccount,
   OUTPUT ocOutputParameters,
   OUTPUT olOperationSuccceeded,
   OUTPUT ocValMsg).

TEMP-TABLE ttAcctGrpGlAccount:WRITE-XML("LONGCHAR",vlcTtAcctGrpGlAccount).
ocOutputParameters = "<![CDATA[" + ocOutputParameters + "]]>".
vlcXmlDoc = "<XMLDOC>" +
        SUBST("<OutputParameters>&1</OutputParameters>",ocOutputParameters) +
        SUBST("<OperationSuccceeded>&1</OperationSuccceeded>",olOperationSuccceeded) +
        SUBST("<ValMsg>&1</ValMsg>",ocValMsg) +
        "<ttAcctGrpGlAccount>" +
        "<![CDATA[" + vlcTtAcctGrpGlAccount + "]]>" +
        "</ttAcctGrpGlAccount>" +
        "</XMLDOC>"
        .
{&OUT-LONG} vlcXmlDoc.
```

Call the server side "presenter" layer.

Translate the outputs into a (nested) XML document

<!CDATA[. .]]> tag can be used to embed an XML document within an XML document

What the output of write-xml looks like

C:\TEMP\ttAcctGrpGlAccount.XML

File Edit View Favorites Tools Help

Address file:///C:/TEMP/ttAcctGrpGlAccount.XML#


```
<?xml version="1.0" ?>
- <ttAcctGrpGlAccount xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  - <ttAcctGrpGlAccountRow>
      <DebitCredit>credit</DebitCredit>
      <Seq>1</Seq>
      <GlAccountNumber>00512002000110000 0</GlAccountNumber>
      <GLAccountDescription>Inventory - Finished Goods - Material</GLAccountDescription>
      <ConfiguredIn>Configured in Account Code # I10</ConfiguredIn>
      <GroupCd />
      <AcctTypeCn />
      <hWidgetHandle xsi:nil="true" />
      <hWidgetHandle xsi:nil="true" />
      <hWidgetHandle xsi:nil="true" />
    </ttAcctGrpGlAccountRow>
  - <ttAcctGrpGlAccountRow>
      <DebitCredit>credit</DebitCredit>
      <Seq>2</Seq>
      <GlAccountNumber>00512002000111000 0</GlAccountNumber>
      <GLAccountDescription>Inventory - Finished Goods - Labor</GLAccountDescription>
      <ConfiguredIn>Configured in Account Code # I11</ConfiguredIn>
      <GroupCd />
      <AcctTypeCn />
      <hWidgetHandle xsi:nil="true" />
      <hWidgetHandle xsi:nil="true" />
      <hWidgetHandle xsi:nil="true" />
    </ttAcctGrpGlAccountRow>
  + <ttAcctGrpGlAccountRow>
```


My Computer

Using the <![CDATA[....]]> tag to embed an XML document within and XML document

http://127.0.0.1/cgi-bin/cgiip.exe/cgiAccountGrpDiag.r?Operation=GetTransactionTypes&InputParam - Microsoft Internet...

File Edit View Favorites Tools Help

Address http://127.0.0.1/cgi-bin/cgiip.exe/cgiAccountGrpDiag.r?Operation=GetTransactionTypes&InputParameters=GroupCd**JT8CC`ViewMode**advancec

```
- <XMLDOC>
  - <OutputParameters>
      <![CDATA[    ]]>
    </OutputParameters>
    <OperationSuccceeded>no</OperationSuccceeded>
    <ValMsg />
  - <ttAcctGrpGlAccount>
      <![CDATA[ <?xml version="1.0"?><ttAcctGrpGlAccount
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><ttAcctGrpGlAccountRow><DebitCredit>credit</De
    </ttAcctGrpGlAccount>
  </XMLDOC>
```

Done Internet

Recap of the layering approach: Things to define

- Define UI in terms of set of “Event/State change” relationships
- Define an API for every “Event/State change” relationship
- Define generic signature for the GetData method

Recap of the layering approach: Sequence of steps

- Create UI
- Create client side presenter layer (class or persistent procedure)
- Create server side presenter layer (appserver procedure)
  - Implements GetData signature
- If necessary create CGI wrapper around server side presenter layer to translate signature into XML

Session : 1017

Title : Future proofing your application through layering

Subtitle : How to support multiple UI platforms simultaneously

Frank Hilhorst
President Progressive Consulting Inc.
Email: Frank@ProgressiveConsultingInc.com
Phone #: 561-8432839

Bonus Slides

Useful ABL/OO features to support layering

- Defining an interface
- Inheritance and overriding
- Overloading

Advantages of defining an interface

- Class signs a contract to support a certain set of methods with a certain input/output signature
- Compliance is verified at compile time

How to define and implement an interface

Procedure Editor - C:\Progress Presentation\Code Extract\CopdeExtract7.p

File Edit Search Buffer Compile Tools Options Help

```
USING Progress.Lang.*.

INTERFACE Cosmos.AccountGrpDiag-inf:
   {AccountCode/AccountGrpDiag.i}
   METHOD PUBLIC VOID GetAccountGroupInfo( icGroupCd AS CHARACTER,
                                           output ocGroupNm AS CHARACTER,
                                           output ocGroupDesc as CHARACTER):
   METHOD PUBLIC VOID GetTransactionGroups(OUTPUT ocListItemPairs AS CHARACTER):
   METHOD PUBLIC VOID GetTransactionTypes(icTransactionGroupCd AS CHARACTER,
                        icViewMode AS CHARACTER,
                        OUTPUT ocListItemPairs AS CHARACTER  ):
   METHOD PUBLIC VOID ProcessTrType(INPUT icGroupCd AS CHARACTER,
                       INPUT icTransactionType AS CHARACTER,
                       OUTPUT TABLE FOR ttAcctGrpGlAccount,
                       OUTPUT ocSubTypeListItemPairs AS CHARACTER,
                       OUTPUT ocOccursWhen AS CHARACTER,
                       OUTPUT ocDocNote AS CHARACTER):
   METHOD PUBLIC VOID ProcessTrSubType(icGroupCd AS CHARACTER,
                         icTransactionSubType AS CHARACTER,
                         OUTPUT TABLE FOR ttAcctGrpGlAccount,
                         OUTPUT ocOccursWhen AS CHARACTER,
                         OUTPUT ocDocNote AS CHARACTER):
END INTERFACE.

CLASS Cosmos.AccountGrpDiag IMPLEMENTS Cosmos.AccountGrpDiag-inf:

END CLASS.
```

Definition of the interface

Implementation of the interface

Use of inheritance in client side presenter class

- Presenter class for web services implementation can inherit from Appserver class
  - Inherits interface methods
  - Overrides GetData method

Definition of “Web Service” client side presenter class using inheritance

Procedure Editor - C:\Progress Presentation\Code Extract\CopdeExtract8.p

File Edit Search Buffer Compile Tools Options Help

```
CLASS Cosmos.AccountGrpDiagCtrl-WS INHERITS AccountGrpDiag IMPLEMENTS AccountGrpDia
  {AccountCode/AccountGrpDiag.i}
  /*------------------------------------------------------------------------
      Purpose:
      Notes:
  ------------------------------------------------------------------------*/

  CONSTRUCTOR PUBLIC AccountGrpDiagCtrl-WS ( ):
    SUPER ().
  END CONSTRUCTOR.

  DESTRUCTOR PUBLIC AccountGrpDiagCtrl-WS ( ):

  END DESTRUCTOR.

  METHOD PROTECTED OVERRIDE VOID GetData(icOperation AS CHARACTER,
                          icInputParameters AS CHARACTER,
                          INPUT-OUTPUT TABLE ttAcctGrpGlAccount,
                          OUTPUT ocOutputParameters AS CHARACTER,
                          OUTPUT olOperationSuccceeded AS LOGICAL,
                          OUTPUT ocValMsg AS CHARACTER ):
    /* Define web services implementation here */
    .........
    return.
  END METHOD.

END CLASS.
```

Use of overloading

- Define the GetData method multiple times with different input/output parameter signatures
  - Allows this method to be called with only the parameters that are relevant for that particular implementation