

Architecting for User Extensibility

Russell Miller
CTO, SunView Software, Inc
@RussMiller123
Russ-Miller.com

About Me

- MS in Computer Science: Northern Illinois University
- Worked for many large software companies (e.g. IBM, Intuit, Quark, Compuware)
- 20+ years architecting software and leading teams
- Co-founded SunView Software 10 years ago
- IT Service Management / Business Process Automation
- Geeking out on Meta for about 15 years (following: Johnson, Yodel, Wirfs-Brock, an others)



What sort of extensibility?

Core extensibility—not just surface level



What's new?

- Customer expectations!!!
- Extensibility itself is not new, what is new is the breath and depth of the expected adaptability.
- Virtualization—cheaper to deal with overhead of meta
- Cloud—scale of economy, large non-captive user base
- Dynamic languages and NoSQL
- Acceptability of separating “what” from “how”

What's not new...

- Too many apps being crafted from hand with hard coded logic and model
- Why? Adaptive Object Models noted over a decade ago?
- It is much easier with current technology and scale of economy is there.
- We need industry wide, open source solution on the level of Open Stack.



Architecture

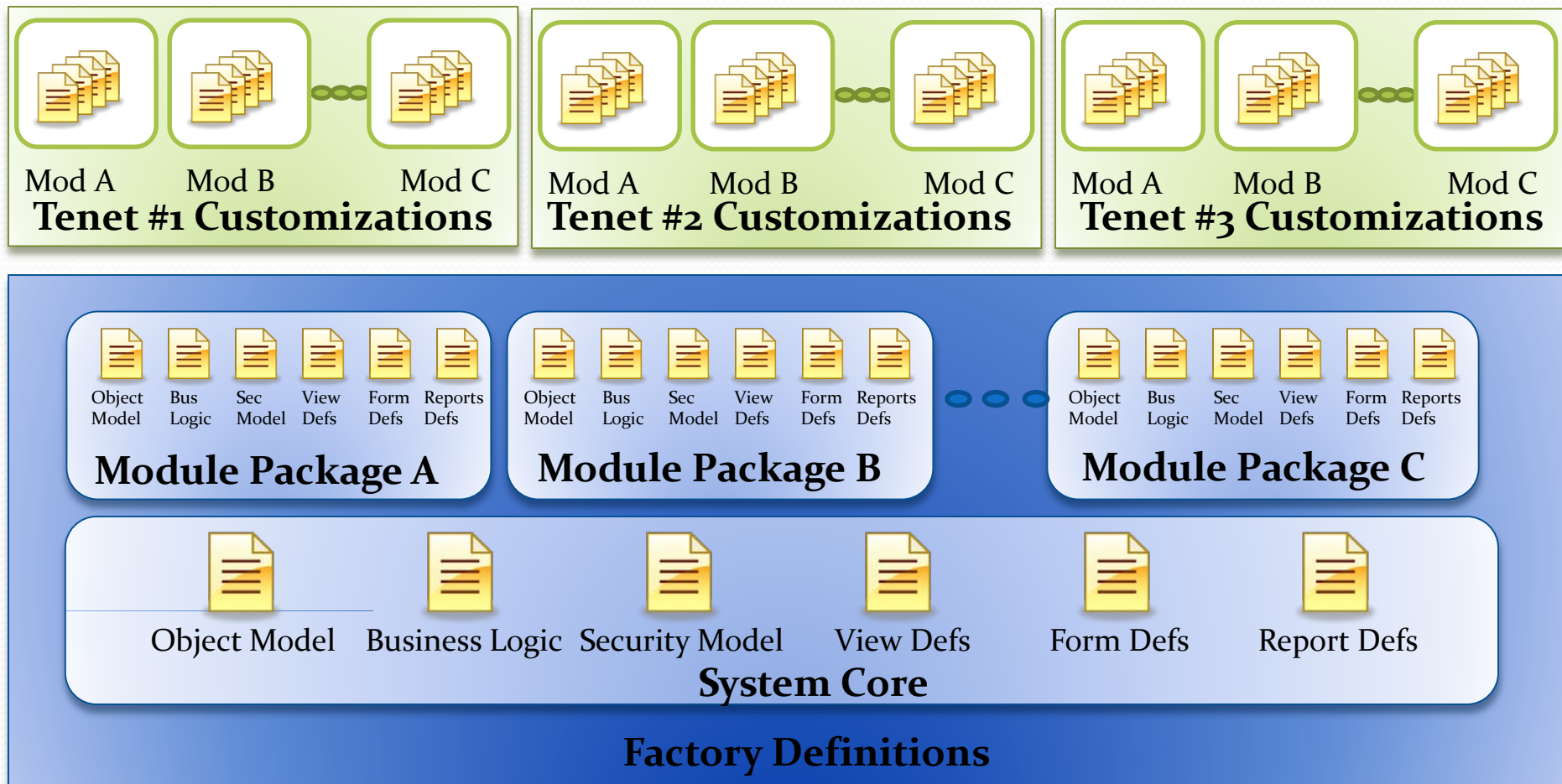
- Don't bake in the “what” and the “how”.
- Separate the “what” from the “how”.
- Make it easy for the customer to evolve both.
- Leverage dynamic nature of latest runtimes

Areas of Extensibility

- Object Model
- Business Logic
- Security Model
- User Actions
- UI Customization
- Reporting
- Integration Interfaces

All of the above, deliverable in a **Packages**

View of the Externalized What/How

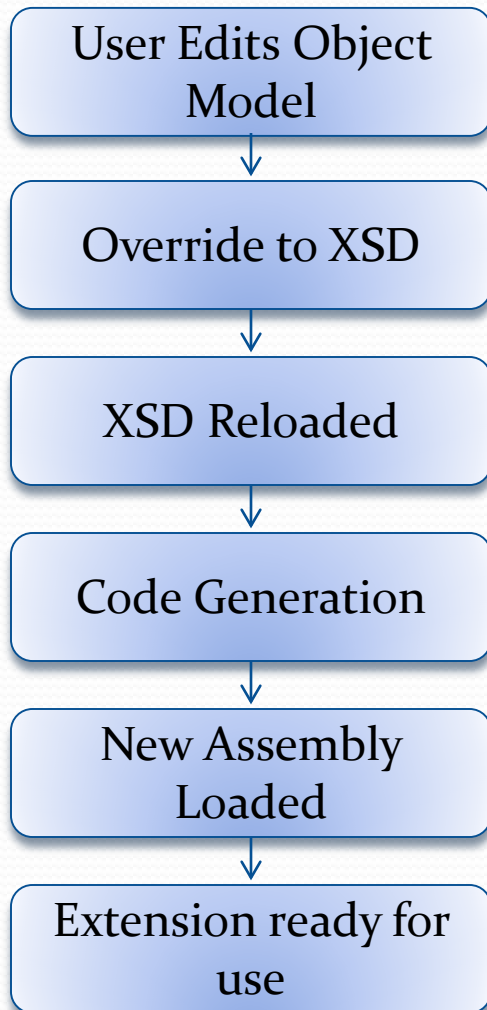


Example – Object Model Edit

The screenshot shows the Entity Editor application with the Configuration Management tab selected. The left pane shows a tree view of modules, with 'Configuration Management' expanded to show 'Cls' > 'Base Cl' > 'Fields' > 'Cost'. The right pane displays the properties for the 'Cost' field.

Configuration Management	
Properties	
Help Revert to factory	
▲ Misc	
▲ FieldAttributes	AllowNull: True, IsRequired: False, Caption: Co
AllowNull	True
CanCopy	True
Caption	Cost
Category	Procurement/Retirement
CopyToTemplate	True
IsCurrency	True
IsReadOnly	False
IsRequired	False
LookupType	
Typeld	<code>OryxAttributes.Attributes.Fields.DecimalFieldAttribute</code>
VisibleToUser	True
FieldDataType	Decimal
FieldName	Cost

Object Model Extension



Override:

```
<xs:element name="Costs" VisibleToUsers="True"
  VisibleForSearch="True"
  IsRequired="False"
  default="" minOccurs="0">
  <xs:simpleType>
    <xs:restriction base="xs:decimal">
      <xs:maxLength value="5.7" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Override approach makes system upgrades possible without losing extensions.

D&D From Object Model to Forms

The image shows a screenshot of the Microsoft Dynamics CRM Form Designer interface. The main window is titled "Form Designer - Default" and contains a menu bar (File, View, Help) and a toolbar with icons for saving, undo, redo, and design mode. Below the toolbar are tabs for "General", "Configuration", "Procurement", "Relationships", "Usage", "Service History", "Network Ports", "Notes", and "Attachments". The "General" tab is active, displaying a form definition with fields for Name, CI #, Type, Class, Criticality, Version, and Description. A "Designer Toolbox" window is overlaid on the bottom right, showing a list of fields and a "Properties" pane. The "Fields" list includes "DeptName", "DeptRootName", "DetailDiffs", "Details", "DisasterPlanning", and "DisasterPlanningRequired". The "Properties" pane shows the following settings for the selected field:

Properties	
Caption	Disaster Planning
CaptionLocati	Top
CaptionSizeM	UseParentOptions
ControlType	Default
DefaultValue	
EnableValueCl	False
FieldName	DisasterPlanning

Imparting the “How”

The image shows a screenshot of a workflow editor interface. The main window is titled "Workflow Details - Change" and has a menu bar with "File", "Edit", "View", and "Help". Below the menu bar is a toolbar with various icons. The main workspace displays a state transition menu with the following items:

- Unsubmitted (with a green circle icon)
- Submit
- Close
- Submit for Approval

Below this menu, there is another menu with a gear icon and the following items:

- New
- Reje
- Esc
- Req
- Def
- Sub

At the bottom, there is another menu with a gear icon and the following items:

- Ne
- Res
- Clos
- Esc

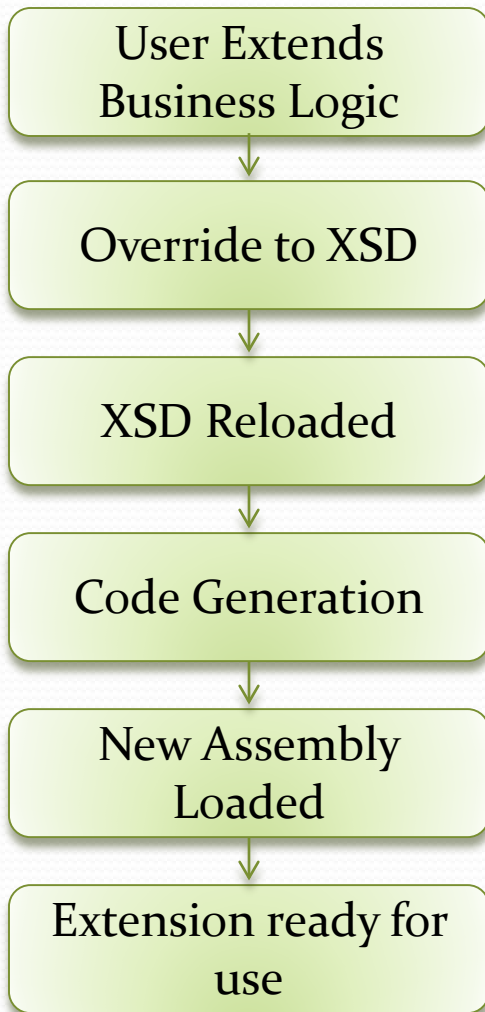
Overlaid on the bottom right is a "Run Code Details" dialog box. It has a title bar and a close button. The dialog is divided into two tabs: "General" and "Conditions". The "General" tab is active and contains the following fields:

- Name:** Calculate time remaining
- Description:** New method to calculate Time Remaining
- Code:**

```
((SDInterfaces.I_UDT_IncidentRequest)
Item).UDF_BusMinRemaining
= (int)((XApplicationInterfaces.Entities.IXApplicationEntity)
Item).CalcBusinessTime(System.DateTime.UtcNow,
((IIncidentRequest)Item).DueDate).TotalMinutes;
```

On the right side of the main window, there is a "Properties" panel with tabs for "States" and "Actions". The "Actions" tab is active, showing "Add", "Delete", and "Show All Actions" buttons. Below these is a dropdown menu set to "Action2". At the bottom of the Properties panel, there are tabs for "Properties", "Rules (0)", and "Automation (1)".

Object Model Extension



Override:

```
<action name="CalculateTimeRemaining" type="Run Code" id="2" Provider="InLineCode"
description="New Method for calculating time remaining" enabled="true">
  <arguments>
    <argument name="Code">
      <codeSegment>
        <![CDATA[(((SDInterfaces.Entities.I_UDT_IncidentRequest_Extended)Item)
        .UDF_BusMinRemaining
        = (int)((XApplicationInterfaces.Entities.IXApplicationEntity)Item)
        .CalcBusinessTime(System.DateTime.UtcNow,
        ((IIncidentRequest)Item).DueDate).TotalMinutes;]]>
      </codeSegment>
    </argument>
  </arguments>
</action>
```

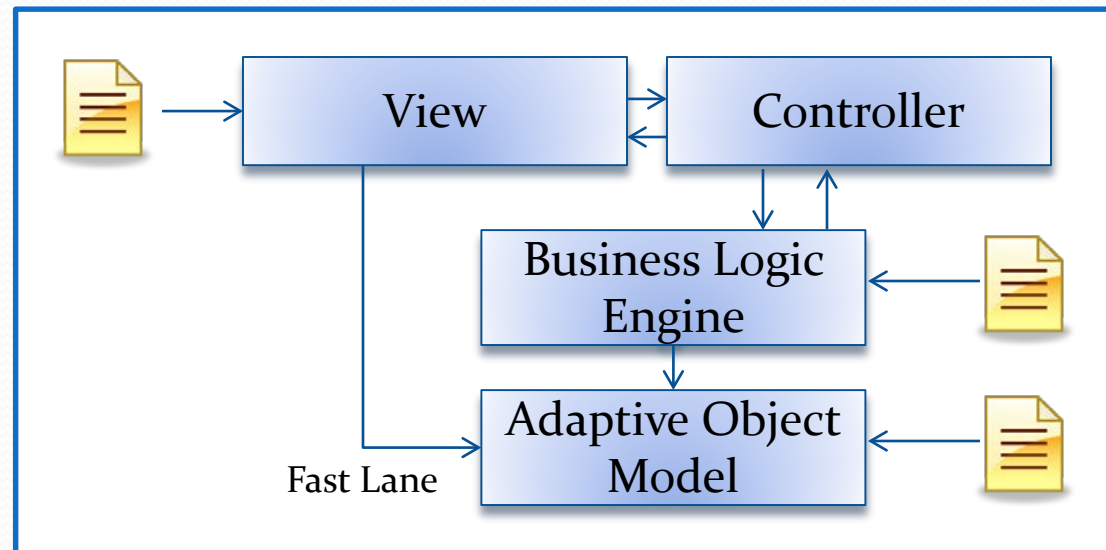
Again: Override approach makes system upgrades possible without losing extensions.

Generic Views and Controllers

Model View Controller

But the View and Controller are “dumb”, very generic.

No hard knowledge of domain above the Business Logic Engine



Separating What from How

- As recognized by Reenskaug and Coplien, very critical to separate “What this system is” from “what it does or how it does it”
- They need to live separate but parallel lives
- Externalized meta-model clearly separates out the “what it is”
- Externalized meta-logic separates out “what it does”
- Looking toward more explicit application of DCI going forward

Future Directions

- Currently uses SQL database, considering NoSQL (polyglot?)
- Currently more pluggable code harder for customers to craft than it should be—DCI exposed for customer?
- Open source community involvement

Conclusion

- Enterprise customers demanding increasing adaptability/extensibility/flexibility
- They expect to adapt at runtime (without an expensive consultant)
- They expect this with no down time (even when upgrading)
- Don't bake in the “what” and “how”—at any level
- Separate the “what” and “how”
- **Where's the industry wide answer? (open source?)**



Questions?

@RussMiller123

Russ-Miller.com