

Build Composite UI Applications with CAB and SCSF

Brian Noyes
Chief Architect, IDesign
(www.idesign.net)

Pre-requisites for this presentation:

- 1) Some Windows Forms Experience
- 2) Understand Reflection and references

Level: Intermediate

About Brian

- | Chief Architect, IDesign Inc. (www.idesign.net)
- | Microsoft Regional Director / MVP
- | Publishing
 - **Developing Applications with Windows Workflow Foundation**, LiveLessons training DVD, June 2007.
 - **Smart Client Deployment with ClickOnce**, Addison Wesley, January 2007
 - **Data Binding in Windows Forms 2.0**, Addison Wesley, January 2006
 - MSDN Magazine, MSDN Online, CoDe Magazine, The Server Side .NET, asp.netPRO, Visual Studio Magazine
- | Speaking
 - Microsoft TechEd US, Europe, Malaysia, Visual Studio Connections, DevTeach, INETA Speakers Bureau, MSDN Webcasts
- | Participates in Microsoft Design Reviews
- | E-mail: brian.noyes@idesign.net
- | Blog: <http://briannoyes.net>

Agenda

- | User Interface Design Challenges
- | What is CAB?
- | CAB Architecture
- | Smart Client Software Factory Overview
- | Building CAB Applications
- | Other CAB Features
- | Composite WPF Applications

UI Design Challenges

- | Repetitive code
 - Main window
 - Top level navigation
 - Look and feel
- | Consistency
 - Construction of forms
 - Event handling
 - View – to – view communications
- | Decoupling
 - View presentation
 - Navigation
 - Communications

Separation of Concerns

- | Code with a different focus or concern should go in a separate logical layer
 - Minimum: Separate class in the same project
 - Better: Separate class library
- | Common layers:
 - Presentation Layer
 - Business Layer
 - Data Layer
- | Layers can also be tiers
 - Potential for physical separation (i.e. other machine)
- | For complex layers, you also need separation of concerns within the layer

Agenda

- | User Interface Design Challenges
- | What is CAB?
- | CAB Architecture
- | Smart Client Software Factory Overview
- | Building CAB Applications
- | Other CAB Features
- | Composite WPF Applications

What is CAB?

- 1 Composite UI Application Block
- 1 Framework for building complex smart client applications
 - Patterns based
- 1 Developed by Microsoft patterns and practices
- 1 Modular approach to building user interface applications
 - Minimizes coupling between use cases
 - Works well for large applications / distributed development teams
- 1 Implements common needs for UI applications
 - Module loading
 - Event publishing
 - Command handling
 - State persistence

Examples: Dell Call Center

The screenshot displays the 'Contact Center Agent Desktop' interface. On the left, there is a 'Session Explorer' pane showing two sessions: 'John Smith - (425) 533-3335' and 'Merve Stodolch - (425) 221-0456'. Below this is a 'Current Workflow (Non-Faced)' pane with a 'Workflow Steps' list including 'Customer's Alerts', 'Latest Promotions', 'Billing Information', and 'Update Customer Information'. The main area shows the 'Update Customer Profile' form for 'John Smith - (425) 533-3335'. The form includes fields for Personal Details (First Name, Middle Initial, Last Name, Gender), Contact Details (Address, City, State, Zip Code, Email, Date of Birth, Contact Number), Billing Information (Billing Address, Billing State, Billing City, Billing Zip Code), and Bank/SSN Information. A 'DELL' logo is visible in the top right corner of the interface.

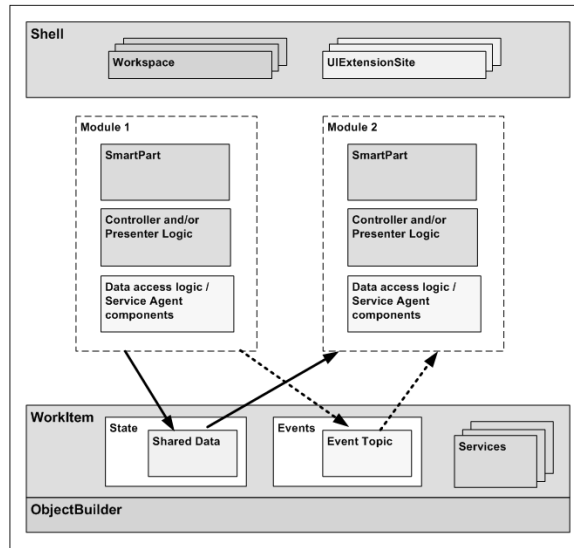
Agenda

- | User Interface Design Challenges
- | What is CAB?
- | CAB Architecture
- | Smart Client Software Factory Overview
- | Building CAB Applications
- | Other CAB Features
- | Composite WPF Applications

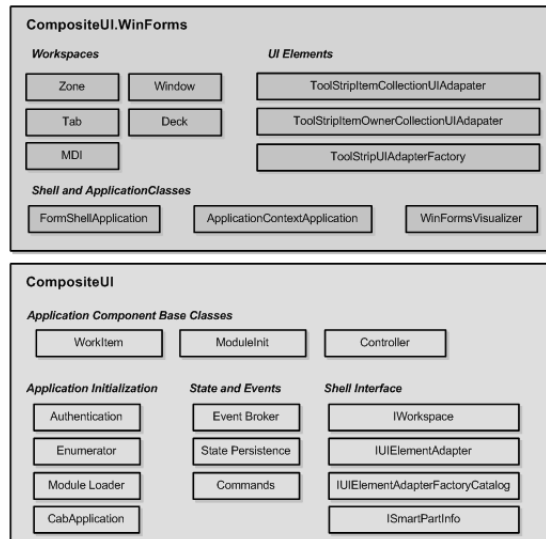
CAB Terminology

- | Application Shell
 - Top level application and main window
- | Module
 - Container for related components
- | WorkItem
 - Container components satisfying a use case
- | Services
 - Shared functionality across modules or workitems
 - Singletons
- | SmartParts (aka View)
 - Modularized UI components - based on UserControl
- | Workspaces
 - UI Containers for controls
- | UI Extension Points
 - UI plug in points - toolbar buttons, menu items, status strip, etc.

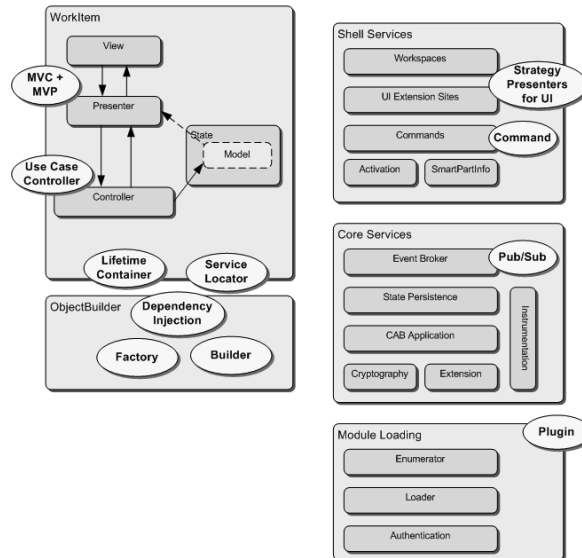
CAB Architecture



CAB Architecture



Patterns Based Architecture



Dependency Injection

- l Also known as IoC – Inversion of Control
- l Pattern that allows indirect construction and initialization of objects
- l CAB Dependency Injection based on attributes
 - ServiceDependency
 - CreateNew
- l Container detects attributes and performs construction and initialization of the appropriate object type
 - Example: Accessing a WorkItem or CAB Service

Agenda

- | **User Interface Design Challenges**
- | **What is CAB?**
- | **CAB Architecture**
- | **Smart Client Software Factory Overview**
- | **Building CAB Applications**
- | **Other CAB Features**
- | **Composite WPF Applications**

What is a Software Factory?

- | **Structured collection of related software assets**
- | **Provides guidance and automation for building applications of a particular type**
- | **Can customize and extend if for a particular organization or team**

Smart Client Software Factory

- | Next step in the CAB evolutionary chain
- | Makes CAB easier
- | Builds on CAB to provide:
 - Architecture guidance and patterns
 - Application blocks
 - Quickstarts
 - Reference implementations - full scale sample app
 - How-to topics
 - Recipes (Guidance automation)
 - Reusable code

Agenda

- | User Interface Design Challenges
- | What is CAB?
- | CAB Architecture
- | Smart Client Software Factory Overview
- | Building CAB Applications
- | Other CAB Features
- | Composite WPF Applications

CAB Design

- I Use Case Analysis
 - Need to do some up front design
 - Partition functionality into use cases
 - Allocate use cases to modules
- I Shell design
 - Decide what the top level layout of the main window is going to be
 - What UI Extension points will the shell provide
 - What common services will be needed by the modules

CAB Development

- I Create a Shell
- I Create a Module
 - Create WorkItem(s)
 - Create a WorkItemController
 - Create Views + Presenters
 - Create Services
 - Implement logic
- I Add module to application
 - Profile catalog
- I Deploy

Customizing the Shell

- I Layout the Shell form
 - Add Workspaces
 - DeckWorkspace**
 - TabWorkspace**
 - MDIWorkspace**
 - WindowWorkspace**
 - ZoneWorkspace**
 - Add UI Extension points
 - Menu**
 - Toolbar**
 - Status bar**

Designing Modules

- I Use SCSF guidance package recipes
 - Business Module
 - Foundational Module
- I Add initialization / business logic to **WorkItemController**
 - Real business logic should be in web services
- I Add Views

Module Initialization Code

- I Construct WorkItems
- I Run controller

WorkItemController Logic

- I Plug in UI Extensions
- I Add Services
- I Add event publications
- I Add event subscriptions
 - Call helper methods, business logic, web services
- I Add command handlers
 - Call helper methods, business logic, web services
- I Show Views

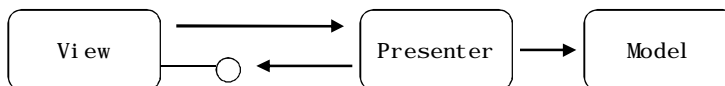
Profile Catalog

- | Controls loading of modules
- | Entry for each module that you want to load
- | Keep Shell decoupled from module definitions
- | Guidance package recipes add module entry when you add to solution containing Shell project

- | Can be accessed remotely
 - Bank Branch Workbench reference application

Designing Views

- | Use Model View Presenter pattern



- | Define Presenter → View interaction through interface
- | Better decoupling
 - Can change views without impacting presentation logic
 - Can test presentation logic without UI

Designing Views

I View Interface

- Specifies the communications contract from the Presenter to the view
- One way contract
 - Communications from View to Presenter based on public API of Presenter**
- Can define separate View -> Presenter interface if desired

Designing Views

I View:

- Implements interface
- Derives from UserControl
- Marked with SmartPart attribute
- Added to WorkItem Views collection through dependency injection
- Creates its presenter through dependency injection attribute (CreateNew)
- Passes a reference to itself to the presenter
- Notifies its presenter when it is ready (Loaded)

Designing Views

- | Presenter:
 - Derives from Presenter<TView>
 - Gets reference to its view interface through base class
 - Set by view on construction**
 - Gets a reference to the WorkItem through dependency injection in the base class
 - Gets to WorkItemController through WorkItem**
 - Where the model lives

Agenda

- | User Interface Design Challenges
- | What is CAB?
- | CAB Architecture
- | Smart Client Software Factory Overview
- | Building CAB Applications
- | Other CAB Features
- | Composite WPF Applications

Event Broker

- I Challenges
 - One-way notifications and events are necessary for UI coordination
 - Publishers and subscribers should not be tightly coupled
 - Could be in separate modules**
- I Solution
 - Event Broker
 - Create EventPublications
 - Events identified with an event URI:
 - event://MyApp.Event1 or just MyEvent**
 - Many publishers, many subscribers
 - Strongly typed event arguments
 - Multithreaded synchronization

Event Broker Features

- I Strong Typing -
 - EventHandler<T> lets you pass your own eventArgs type (e.g. MyType : EventArgs)

```
[EventSubscription("event://foo",EventScope.Global)]
public void DoSomething(object source, MyType stuff)
{ ... }
```
- I Subscribe to receive on different Threads
 - Using the Background worker:

```
[EventSubscription("event://foo/bar/baz",EventScope.Global,isBackground=true)]
```

Event Broker Features

- I Event Scoping
 - Global Events and WorkItem-Wide Events
 - [EventSubscription("event://foo/bar/baz",EventScope.Global)]
 - [EventSubscription("event://foo/bar/baz",EventScope.WorkItem)]
- I Works on components, normal classes and static types
 - Components and objects get "inspected" when added to a Context or Host
 - Static types need to be manually registered
- I Can publish, subscribe to, and fire events programmatically as well as declaratively

Commands

- I Many UI elements can trigger the same action
 - Menu item, toolbar button
- I Need to enable/disable UI elements in unison
 - And not fire the events if disabled
- I Commands are named actions that can have multiple source elements that trigger it and multiple handlers that handle the event
- I Declarative hook up

Agenda

- | User Interface Design Challenges
- | What is CAB?
- | CAB Architecture
- | Smart Client Software Factory Overview
- | Building CAB Applications
- | Other CAB Features
- | Composite WPF Applications

What About WPF?

- | SCSF 3.0 supports WPF Views
- | CAB WPF Community project
- | Prism
 - New p&p project
 - Readdresses the problem from scratch
 - Takes capabilities of WPF into better account
 - Lighter weight than CAB/SCSF
 - First release later this year
 - Reference Implementation + Guidance (docs)
 - <http://www.codeplex.com/prism>

Resources

- | Programming Microsoft Composite UI Application Block and Smart Client Software Factory, David Platt, Microsoft Press, 2007.
<http://www.microsoft.com/MSPress/books/11030.aspx>
- | Architecting Composite Smart Clients Using CAB and SCSF, Mario Szpuszta, The Architecture Journal, Jan 2007.
<http://msdn2.microsoft.com/en-us/library/bb266334.aspx>
- | Smart Client Guidance –
<http://www.codeplex.com/smartclient>
- | Prism
<http://www.codeplex.com/prism>

E-mail: brian.noyes@i design.net
Blog: <http://briannoyes.net>