Workflow Design for a Web Commerce Application

Duwamish Books, Phase 4

Steve Kirk
Microsoft Developer Network

August 1999

Summary: Describes the workflow layer used in Phase 4 of the Duwamish Books sample application. (11 printed pages)

Introduction

The workflow layer is implemented as a COM component combined with Active Server Pages (ASP) script code to provide a common set of XML interfaces for the three client applications included in Duwamish Books, Phase 4. The Phase 4 client application types include a simple HTML 3.2 client; a Microsoft® Internet Explorer 4.0 client, which uses data binding with a client-side COM data source object; and an Internet Explorer 5 client that consumes XML directly.

This workflow server, based on COM components running under Microsoft Transaction Server (MTS), and exposed to the Internet through ASP pages, can easily scale to serve a high-volume e-commerce business and, because it serves up XML, can work with any presentation technology and easily interoperate with other applications.

For a thorough comparison of the three client types, see "Three Approaches for Three Client Types." The XML interfaces exposed by the component are detailed in "The Duwamish Books, Phase 4 Workflow API Reference."

The following discussions assume that you are familiar with the Windows DNA architecture and the layered architectural paradigm discussed in "An Introduction to the Duwamish Books Sample," which covers the terms data access layer, business logic layer , workflow layer, and presentation layer. The Duwamish Books, Phase 4 Workflow (WFL) COM component is a client of the Business Logic Layer (BLL) COM component running under MTS. See "The Duwamish Books, Phase 4 Business Logic Layer API Reference" and "Abstracting Business Transactions" for more information on the BLL component.

So, Exactly What do you Mean by "Workflow"?

By workflow I mean logic that coordinates smaller pieces of work to accomplish a larger, more complex task. Workflow logic differs from transactional logic in its tolerance for incomplete work. A transaction seeks to be either complete, where all pieces of work become permanent, or completely undone. Workflow logic, in contrast, allows many sessions to remain at an incomplete state for a comparably long period of time without incurring significant overhead. In Duwamish Books, Phase 4, workflow controls the shopping experience for a large number of shopping sessions, even though relatively few will result in completed sales. (See "Setting Performance Goals for Duwamish Books, Phase 4" for more on the relationship between searches and purchases in an e-commerce application.)

Figure 1 shows the logical workflow layer with the presentation and business logic application layers. To the client-side of the workflow layer is the presentation layer, which contains logic specific to a presentation technology. To the server-side of the workflow layer is the business logic layer, which simplifies business transactions.

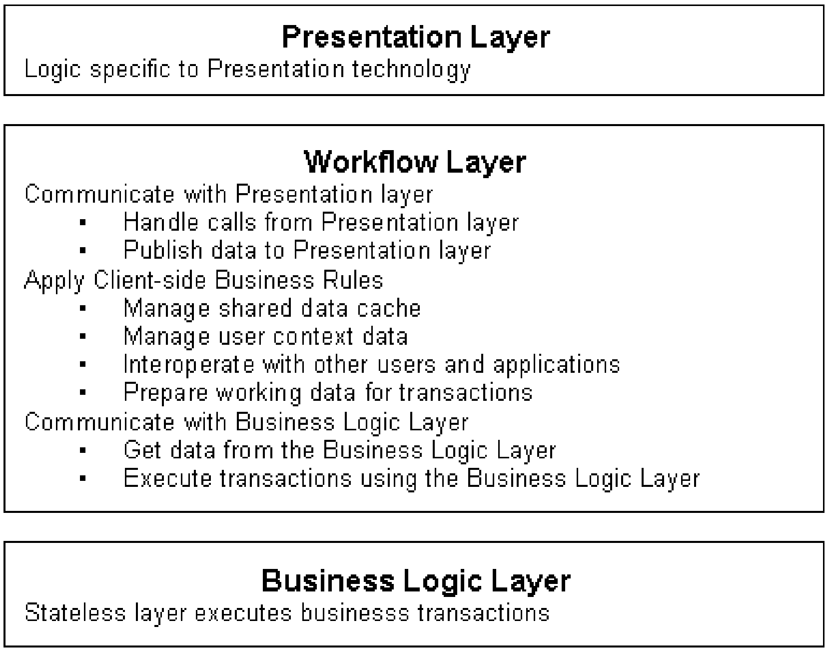

Figure 1. Application logic in the Duwamish Books sample

Workflow and the Presentation Layer

The presentation layer is comprised of the user interface and technology-specific logic that maps the UI to the workflow layer. Presentation layer logic translates user input such as a mouse click or keystroke, into a workflow method such as adding an item to a shopping cart. In the other direction, workflow logic drives the UI through virtual methods exposed by the Presentation layer, such as displaying an item.

Workflow and the Business Logic Layer

In Duwamish Books, Phase 4 business rules are divided into two complementary layers: workflow and business logic. The workflow layer is closer to the client and it is stateful, which means that it holds data between method calls. The stateful workflow layer accumulates contextual data (or state) and executes methods on the business logic layer. The business logic layer is the more server-side of the two, and is stateless. The stateless business logic layer design simplifies resource sharing and promotes scalability by allowing a single BLL instance to be shared by multiple WFL instances without restoring context for each instance. BLL servers may employ load balancing or resource pooling to achieve scalability.

A commerce application provides the user with ways to find merchandise, assemble the merchandise with customer and shipping information into an order, and execute a transaction that saves the order. Workflow logic controls assembly of accumulated data into a potential order as the customer shops and selects merchandise. Eventually, after client-side business rules have been applied and the order is ready to be executed, workflow logic executes a single method on the stateless BLL, passing the packaged order data, to execute the transaction.

The stateful workflow layer works with the stateless business logic layer to achieve important scalability and usability advantages. The WFL does its transactional work through the BLL and effectively releases BLL instances between method calls. Because the transactional resources are acquired through the BLL, the number of open transactional resources does not grow directly with the number of simultaneous workflow instances. This decoupling between layers allows the application to use resources effectively.

The BLL simplifies business transactions so that workflow code does not have to be concerned with implementation details. This allows workflow code to execute transactions, like **InsertSale,** without concern for implementation details beyond the BLL interface. The BLL will either execute the transaction and report success or it will fail and report the reason to the WFL.

Workflow and Caching

In Duwamish Books, Phase 4, workflow logic controls use of a cache to enhance application scalability and usability. Use of a cache at the workflow layer eliminates redundant trips to the back-end database to fetch infrequently changing data and eliminates redundant formatting of that data for the presentation layer. Caching enhances usability by shortening the response lag as the user navigates through the application. Plus, data that is cached close to the presentation layer in a format that is understood by the presentation layer can easily provide twenty times the performance of an application that retrieves data from the database and formats that data on each request (see "Performance Testing a Scalable Application.").In a Web commerce application this performance gain is effectively multiplied because it speeds up catalog browsing, which is a statistically large (80 percent, by our estimate) component of total server requests on commerce sites. For more information on estimated request types, see "Setting Performance Goals for Duwamish Books, Phase 4."

In client type 2 of Phase 4, the local cache is pushed out to the client machine. The next logical step (left to the reader) is to persist the cache on the client machine to enable a disconnected scenario where the user works offline and connects intermittently to add to the local cache or to submit orders as required. Of the many ways available to persist user state, simply writing the XML to a disk file may be the most useful because it follows the widely understood document paradigm.

For more details about the Duwamish Books caching strategy, see the article "Creating a Page Cache Object."

Workflow and User Context

User context is the set of data that accumulates as the user works. An example of user context in a commerce application is a shopping cart. When the user selects a merchandise item and retrieves a description of the item, the application provides a way to add the item to the shopping cart. The contents of the shopping cart are a part of the user's contextual data and the **AddToShoppingCart** method is one of the workflow methods that operate on the user's contextual data. User context may be thought of as a document that can be saved or transmitted, or as a virtual document comprised of data stored in multiple places.

User context on the client

One possible location for user context is on the client. Storing user context on the client reduces server load but requires a client capable of holding context (for example, in a cookie) and may require the user to explicitly accept storage of this data. If user context is written to a file or other persistent storage, the user can shut down the computer or disconnect from the network and continue working from the saved data. Another scenario is when the context "document" is sent to another user who can add to or review the first user’s work. (This concept is the more widely used meaning of the term "workflow.")

User context on a middle tier

Another location for contextual data is on a middle tier. The advantage of storing contextual data away from the client is that it becomes possible for the user to move from one computer to another, reestablish context, and continue with work in progress. However, middle-tier storage of user context means that load-balancing systems have to return the user to the same middle-tier server where the context is stored, obviously reducing the effectiveness of the load balancing.

User context in the database

User context can also be stored in the back-end database where it’s equally accessible from any middle-tier computer. Back-end context storage has the advantage of not suffering from the server affinity experienced in the middle-tier option.

Disadvantages of back-end context storage are the database resources required and the computing cost of frequently moving the data from client to back end.

User context in Duwamish Books

Use of XML for context data in the Duwamish Books, Phase 4 workflow layer simplifies storage and retrieval in an implementation-independent manner. Implementation differences are hidden behind the abstraction of **GetContextData** and **PutContextData**, which means that any of the preceding methods for storing user context can be implemented in Duwamish Books by simply modifying these two methods.

Workflow as Client-Side Business Logic

Workflow code applies client-side business logic to user input and, by controlling the presentation layer, guides the user toward completion of business transactions. The workflow layer validates user input and applies client-side business rules as part of the process of preparing user work for submission, to the (transactional) business logic layer. Although the transactional or server-side BLL component has the last word on what data will be accepted, client-side input validation increases transactional scalability of the system by reducing the distance the data needs to travel to be validated.

As validation is pushed toward the client it becomes possible to increase the amount of validation being done without incurring objectionable computing costs. This can provide better user feedback including immediate input validation and contextual cues.

Implementing Workflow in Duwamish Books

Each of the three client types that comprise Phase 4 requires some workflow code to be implemented as script while common workflow code is implemented in the Workflow (WFL) COM component. Workflow code implemented as server-side script (on an ASP page) in one application is implemented in client-side script in another. A single COM workflow component containing all of the workflow code required by the three applications would not be the best solution because it would require knowledge of specific presentation technologies to be included in the shared code and hinder migration to new presentation technologies.

Code should be located where it makes most sense, and variance between logical function and physical location is often necessary to make best use of technologies. It is important to remember when factoring an application into distinct logical layers that we are not determining a physical location for code. You may remember that we have a similar variance between logical and physical organization in the business logic layer where code in the BLL component calls stored procedures located in the database. The stored procedures implement business logic but provide important performance advantages by being located in the database rather than as code in the BLL component. The Phase 3.5 article "Abstracting Business Transactions" is still very relevant to the Phase 4 implementation, especially in its discussion of these stored procedures.

The Workflow Component and XML Interfaces

The Workflow Layer (WFL) COM component provides an XML interface as a wrapper around the Business Logic Layer (BLL) component. This allows workflow script, as a client of the WFL component, to iterate through XML instead of ADO Recordsets and simplifies transformation for presentation. The XML format also simplifies caching and state management because XML data, as a string, can be cached and passed anywhere a string can be cached.

The following code excerpt from the WFL component **GetItems()** method shows how workflow code wraps the BLL component **GetItems()** method and returns an XML representation of the items. Note that I've omitted parameters from **GetItems** for clarity:

```
Public Function GetItems(Optional ByVal PKid As Long) As String
Dim oRs As ADODB.Recordset
?
' Call the BLL GetItems method to populate recordset
oBll.GetItems oRs, PKid, ?
?
' Convert resulting recordset to XML
sItems = XMLOut(oRs, scITEM, icATTRIBUTED)
?
' Add opening and closing tags and line breaks to Format the XML
GetItems = scTAG_ITEMS & vbCr & sItems & scENDTAG_ITEMS
?
End Function
```

As you can see, I use a function called **XMLOut()** to convert data in the ADO Recordset to XML. I wrote this function instead of using the ADO 2.1 **Recordset.Save** method with the XML option or the

Internet Explorer 5 XML Document Object Model (XMLDOM). The ADO **Save** method doesn't allow the level of control of XML formatting that I wanted. And while the XMLDOM provides near-infinite control, I went with string concatenation because it is a simple solution. The string concatenation I used in **XMLOut()** is possible because the pieces of XML that I am generating are fairly small. You may want to consider other options when working with larger amounts of data.

The XML returned by **GetItems** looks like the following:

```
<Item PKId="789" PublicationYear="1993" ISBN="6-775-47371-6" ImageFileSpec="TX0167.g
```

For a the full documentation of the Workflow Layer component interfaces, see "The Duwamish Books, Phase 4 Workflow API Reference."

Workflow Script and the Presentation Layer

As I stated earlier, the Workflow Layer manages the conversation between the presentation layer and the workflow Layer.

Although all three client types use the WFL component, workflow methods are not all implemented in the WFL. Workflow methods are implemented in a combination of ASP script, client-side script, and a client-side ActiveX® component depending on the client type. In the user input direction of this conversation, presentation logic translates user input into workflow methods. I've paraphrased client-side presentation script from default.htm in client type 3 to show how a user's click on a book in a list is translated into a workflow method call:

```
Function divResLst_OnClick()
?
' Get the ItemID from the HTML element that was clicked
ItemId = Window.Event.SrcElement.GetAttribute("ItemId")
?
' Prepare XMLDOM objects to hold XML description of Item and XSL
XMLItemDetDoc.async = false
XMLItemDetDoc.load("xmldet.asp?" & ItemId)
XSLDoc.async = false
XSLDoc.load("det.xsl")
?
' Execute a Workflow method to get XML representation of Item
' load XML into an XMLDOM object
XMLItemDetDoc.load("xmldet.asp?" & ItemId)
?
' use XMLDOM to transform XML into HTML using XSL
HTMLItem = XMLItemDetDoc.documentElement.transformNode (XSLDoc.documentElement)
?
' Display HTML by assigning entity innerHTML property
divItemDetail.innerHTML = GetItemHTML(ItemId)
?
End Sub
```

For a comparison of the three Phase 4 client types, including more examples of communication between the workflow and presentation layers, see "Three Approaches for Three Client Types."

Implementing the Cache

All three approaches use a cache in the ASP Application object containing category lists and item details. A basic difference between the three approaches is that we cache HTML in client type 1, XML in client type 3, and a combination in client type 2. Although we may cache XML on the server with one client type and HTML on the server with another, the cache control logic is the same. The following workflow code shows the cache logic in use throughout Phase 4:

```
<%
?
' Translate an ID into a cache key.
  DetKey = IdToDetKey(DetId)
' Try to retrieve the data from the cache
DetXML = Application("Cache")(DetKey)
?
' If data was not in cache, get it from the BLL through the WFL
  If DetXML = "" Then
' Instantiate WFL component object
Set WFL = Server.CreateObject("d4Wflow.cWorkflow")
' Call WFL component method to get XML representation of Item
    DetXML = WFL.GetItemByItemId(DetId)
' Add data to cache
Application("Cache").Add DetKey, DetXML
End If
' Write XML data to client as UNICODE
Response.BinaryWrite(Chrw(&HFEFF) & DetXML)
' The function that converts an ID to a cache detail key
Function IdToDetKey(ByVal id)
  Dim s
  s = Cstr(Id)
  IdToDetKey = "Det" & String(9 - len(s), "0") & s
  End Function
%>
```

For a complete description of the Cache component (d4cache.dll), see "Creating a Page Cache Object."

User Context as an XML Document

Each shopping session has a body of contextual data that accumulates as the user shops. This user context, or state, can be stored in many places. It's what must be maintained between HTTP requests to the various Web pages that make up the Web site. If this context is stored as an XML document, it can be stored (as a string) wherever makes sense.

The following XML document contains contextual data for a shopping session where the user has added two books to the shopping cart and, at the checkout, established a customer account and entered a shipping address.

```
<Customer Alias="555-1212" CustomerName="Sear Wellreader" />
<Shipping Address="900 Fake Street YourTown, MN 45678" Country="USA " />
<ShoppingCart>
<Item ItemId="953" Title="Cannibals and Kings" UnitPrice="4.99" Qty="1"/>
<Item ItemId="789" Title="Craftsman's Handbook, The" UnitPrice="14.99" Qty="1"/>
</ShoppingCart>
```

Internet Explorer 5 provides the XMLDOM object model for navigating and manipulating an XML document. We use the XMLDOM object model extensively to manipulate XML throughout Duwamish Books, Phase 4. The following code excerpts from **AddToShoppingCart()** show how an item is added to the shopping cart. The first step (after instantiating the XMLDOM and loading the shopping cart XML) is to determine if the Item is already in the shopping cart and, if so, to increment the quantity:

```
' Instantiate XMLDOM object and load XML shopping cart.
Set XMLDoc = Server.CreateObject("Microsoft.XMLDOM")
XMLDoc.async = False
XMLDoc.loadXML (XMLShoppingCart)

' Iterate through shopping cart looking for ItemId
For n = 0 To XMLDoc.documentElement.childNodes.length - 1
Set XMlElement = XMLDoc.documentElement.childNodes(n)
' If ItemID is found, increment quantity and exit
  If XMlElement.getAttribute("ItemId") = ItemId Then
    XMlElement.setAttribute "Qty", _
      CStr(Clng(XMlElement.getAttribute("Qty"))+ 1)
' Return resulting XML representation of shopping cart
    AddToShoppingCart = XMLDoc.documentElement.xml
' Clean up and exit
    Set XMlElement = Nothing
    Set XMLDoc = Nothing
    Exit Function
  End If
Next
```

The following code executes in cases where the Item is not found in the shopping cart. It calls **GetItemXML**, which manages a local cache of Item data to retrieve an XML representation of an Item for an ItemId. This XML item is loaded into an XMLDOM instance so that its attributes can be retrieved easily:

```
 ' First get an XML representation of item.
XMLItem = GetItemXML(ItemId)
' Instantiate XMLDOM object and load Item XML
Set XMLItemDoc = Server.CreateObject("Microsoft.XMLDOM")
XMLItemDoc.async = False
XMLItemDoc.loadXML (XMLItem)
```

This code creates a new Item in the shopping cart and copies attributes from the Item just retrieved. Finally, the function returns the resulting XML representation of the shopping cart:

```
Set XMlElement = XMLDoc.createElement("Item")
' Set the ItemId, Title and UnitPrice attributes
XMlElement.setAttribute "ItemId", ItemId
XMlElement.setAttribute "Title", XMLItemDoc.documentElement.getAttribute("Title")
XMlElement.setAttribute "UnitPrice", XMLItemDoc.documentElement.getAttribute("UnitPr
XMlElement.setAttribute "Qty", "1"
' Add the Item to the shopping cart
XMLDoc.documentElement.appendChild XMlElement

' Return resulting XML representation of shopping cart
AddToShoppingCart = XMLDoc.documentElement.xml
```

Conclusion

Workflow provides both the macro logic that connects a string of essentially atomic pieces of work into a larger task, and it also manages the user state that is necessary to connect the pieces as a whole.

Using XML as the data format for this layer provides several advantages. XML data can be easily cached, it can be easily transformed for presentation on various technologies, and XML data is readily used by other applications allowing standards-based interoperability.

Finally, maintenance of contextual data as an XML document allows context storage to be handled in an implementation-independent manner. Contextual data can be stored as a string of XML on the client, the middle tier, in the back-end database, or in other locations, and the storage implementation can be easily changed without affecting workflow logic if context storage and retrieval is hidden behind an abstract interface.

Comments? We welcome your feedback at duwamish@microsoft.com.