Developing the Reference Application Pack for the Loan-Origination Systems (OR-LOS)

Mike Walker
Microsoft® Corporation
Subir Talukder
April 2007

Applies to:
Microsoft Visual Studio® development system
Office Business Applications

Summary: The purpose of this article is to walk through the development aspects of the Office Reference Application Pack (RAP) for the Loan-Origination Systems (OR-LOS) solution. Developers and architects who seek more detail on how to build Office Business Applications (OBAs) for the financial-services industry will find this article very useful. Specifically, if you are planning on building a loan-origination system (LOS) for your lending institution, this article will help you make the difficult micro-level design decisions. Although this is only one way of developing a specific architecture for a specific application, we feel that most developers will get valuable insights from the process and the technologies that we employed in the OR-LOS architecture. (56 printed pages)
Introduction

With multiple forces and constraints at work, building a lending system can be a very challenging task. What makes loan-origination solutions so difficult to build is their strict requirements around interoperability with internal and external systems; scaling to unknown volumes that can fluctuate at a moment’s notice; and, finally, securing customer information and other compliance requirements.

This article discusses two aspects of the architecture:

- **Lending framework.** The Reference Application Pack (RAP) is based on a lending framework for loan-origination system (LOS). It provides a highly scalable framework that can be used to unify people, processes, and technologies across organizational barriers. The lending framework allows developers and architects to use this as a platform for many different lending channels. Throughout this article, the lender framework and the lending scenario are discussed in parallel.

- **Lending scenario.** The lending scenario is built on top of the lending framework. This provides a real-world scenario that developers can reuse to jump-start development efforts.

Starting the Development of the OR-LOS Solution

Before we began to develop the Office Reference Application Pack (RAP) for the Loan-Origination Systems (OR-LOS) solution, we needed to understand how we would develop, with which resources, when it was due, and what the specific business needs were.

We started out with an informal requirements-gathering process. Instead of going through great detail around user interface (UI) requirements or business rules, we decided to focus on the key business capabilities we wanted to expose. This is not to say there were no requirements gathered—just not the 100-plus-page documents that you traditionally see in these types of efforts. For example, long-running workflow or approval processes were identified as capabilities. As soon as we identified these capabilities, we started the architecture process.

This type of development is typically called an agile-based development style. Having a core set of experts around a subject matter enabled us to develop this application more quickly. If we did not have subject-matter experts who were highly technical and could translate between business and technology capabilities, this approach would not have worked as well. However, the downside to this approach is that it is easy to miss requirements by your business users.

It is important to understand that when we built this reference application pack, it was meant to be somewhat ambiguous in nature for the audience. Your processes will most likely be very different, based on your organizational and business needs.

Here are the steps we took to build out the OR-LOS solution:

- **Understand the business requirements and process.** Identify the business process. Map the business capabilities to technical capabilities that our architecture exposes.

- **Create an architecture.** Build a solid foundation in which our developers can build the composite application components.
• **Build out functional areas iteratively.** As soon as we had the scenarios and the architecture in place, we had to decide how to build the functionality. We decided to go for an iterative mode of development. So, instead of building all the functionality at once, we decided to build it in little chunks. To give an idea, we started with an initial schema for our Microsoft® Office InfoPath® 2007 template. However, during the course of development, we felt the need to modify the schema. So, we went back and updated the schema and the form template. Then we could use the updated schema in the Office InfoPath form template. Because the modifications were small, they did not affect the development schedule. However, if the changes are significant, it would be necessary to cost the effort accordingly. As soon as we had the mode of development in mind, we had to determine the order of development. For this, we did the following:

  • **Determine dependencies.** We first determined what pieces could be built without having any dependencies. This led us to start from the database model.

  • **Determine core.** As soon as we had the database in place, we had to make a choice about which scenario to work on. We realized that the registration was one of the core scenarios, because it involved a lot of things that we were planning to highlight. So, we decided to build that piece first.

  • **Start the development inside out.** In determining the data model, one of the key inputs was the Mortgage Industry Standards Maintenance Organization (MISMO) standard. We knew that we could not tweak with that standard. So, we modeled the database so that the fields are consistent with the elements in the schema.

  • **Focus on reusability.** Throughout the development, the focus was on using and creating reusable components. For this purpose, we decided to use Microsoft Enterprise Library for our data access and caching support. Similarly, the code to read OpenXml documents was packaged in its own library.

  • **Repeat process for each scenario.** As soon as we had created the registration scenario, we decided to apply what we had learned and the architecture to build out the other scenarios.

  • **Ensure consistency.** We ensured that namespaces and class names are consistently named across the solution.

**What Tools Do I Need?**

The OR-LOS solution uses multiple products for different purposes. The majority of the time, we used Microsoft Visual Studio® development system, but other tools, such as Microsoft Office SharePoint® Designer 2007 or Office InfoPath Designer, were also used as necessary—for example, Microsoft Office SharePoint Server for the lender portal, Microsoft BizTalk® Server for integration, Office InfoPath for creating forms, and so on. So, we needed to use the design tools for each product. Apart from that we also needed to administer the products themselves. This meant that we had to use the administration utilities for each product. One of the important aids for the development process is the various extensions to add-ons available for Visual Studio. By using these extensions, we were able to have a single integrated development environment (IDE), Visual Studio, instead of having multiple development tools. One of the key features these extensions typically provide is the ability to deploy the artifacts to the server or host application. In some cases, this was done through template scripts that could be executed to perform the deployment (for example, in Office SharePoint Server workflows) and in others, the deployment feature was part of the commands that could be executed (for example BizTalk designer). Of course, deploying to actual production/staging/test environments will require a much more sophisticated design.
Development Tools

The development tools used for the OR-LOS solution are:

- **Visual Studio 2005**. Used as the primary design and development environment. Used for developing .NET assemblies, custom Office SharePoint Server workflows, custom Web parts, BizTalk orchestrations, Office InfoPath code-behind. We also used the following plug-ins:
  - Microsoft Visual Studio 2005 Tools for Applications
  - Microsoft Visual Studio 2005 Tools for the Microsoft Office System
  - Microsoft Windows® Workflow Foundation (WF) Extensions for Visual Studio
  - Microsoft BizTalk Server 2006
  - Microsoft SQL Server® Reporting Services Designers
- **Office InfoPath Form Designer**. Used for designing and publishing Office InfoPath forms.
- **SharePoint Designer**. Used for creating and designing Office SharePoint Server sites.
- **BizTalk Rules Composer**. Used for creating business rules that can be used from BizTalk orchestrations.
- **BizTalk Server Administrator**. Used for creating and configuring BizTalk applications.
- **BizTalk Server Configuration**. Used for configuring BizTalk settings.
- **SQL Server Management Studio**. Used for creating and configuring database objects.

Project-Management Tools

We used Office SharePoint Server as the primary tool for collaboration during the development of the solution. In particular, we used the following features:

- **Tasks lists**. For tracking tasks assigned to team members
- **Document libraries**. For uploading documents like status reports, design specifications, and schemas
- **Lists**. For tracking open issues

Technical Design of the Solution

The OR-LOS solution is created as a composite application. This means the solution is a collection of software assets that have been assembled to provide the required business capabilities. These assets are artifacts that can be deployed independently, enable composition, and leverage specific platform capabilities.

Guiding Principles

Throughout the development process, we had the following key principles in mind:

1. **Enterprise application**. The application should adhere to key requirements of being an enterprise-level application. For example, the architecture should ensure that the implementation of public processes is cleanly separated from the implementation of private processes. This allows private
Developing the Reference Application Pack for the Loan-Origination Systems (OR-LOS) processes to change without affecting integrations with trading partners. Also, the architecture should avoid hard-coding of policies, configuration, rules, or any other kinds of business metadata.

2. **Must scale.** The application should be built with scalability in mind. For example, asynchronous message processing should be used wherever feasible.

3. **Secure.** By using Office SharePoint Server as the application layer, we made sure that users are securely authenticated before accessing the system. By creating Microsoft Active Directory® directory services roles and mapping those roles in Office SharePoint Server, we were able to provide role-based authorization for the application.

4. **Highly reliable.** By using workflows within Office SharePoint Server, we were able to leverage built-in features like workflow persistence, tracking, logging, and so forth. By using BizTalk as the integration layer, we were able to make use of the native retry support in BizTalk Send ports.

**Design Decisions**

This section outlines the design decisions we made when developing the OR-LOS solution:

- **Using predefined application blocks from Microsoft patterns & practices.** We decided to leverage the features provided by the Microsoft Enterprise Library. The benefits of using the application block versus a writing custom solution are that:
  - We can use tried and tested code instead of creating new code.
  - We can ensure the solution uses a consistent interface for functions like data access and caching.
  - We can save development effort.

- **Using Office SharePoint Server versus Web applications.** We decided to go with the option of using the workflow support of Office SharePoint Server, instead of hosting a custom Web application. The main reason was to leverage the services provided by Office SharePoint Server—for example, persistence, tracking, and logging.

- **Single sign on.** The solution uses Active Directory integration with Office SharePoint Server for all authentication needs. Because Office SharePoint Server can use Integrated Web Authentication, we can provide a seamless authentication experience. Also, user roles are stored with Active Directory directory services and mapped in Office SharePoint Server. In this way, there is no need to maintain a user-role mapping within the application itself.

- **Performance (persisted XML Files, BizTalk Optimization).** In OR-LOS, we needed to read product data from a data store. Because this data is fairly static, we decided to persist it in an XML file, instead of reading it from the database. The XML file contents are themselves cached using Enterprise Cache Library.

- **Office InfoPath versus Web forms.** We chose Office InfoPath and Form Services, instead of building custom Web pages, because:
  - There is native support for binding XML schema to page fields.
  - Dynamic UI is possible by creating multiple views of the same form. So, there is no need to create multiple pages, which also avoids handling state issues.
• After filling in a form, the data can be automatically stored in the form library as XML. This provides a simple integration with the loan workflow.

• With Office InfoPath, the user experience is more user entry form-centric.

• Web parts (BDC). We decided to use BDC to implement the pipeline summary view, instead of creating a custom Web part, because of its built-in functionality for searching and filtering data. Also, using BDC provides us with a layer of abstraction between the summary and the data store.

• Products and pricing. The OpenXml APIs are used, instead of Microsoft Office Excel® Services API, for the following reasons:
  • We are mainly interested in the values in the file; thus, there is no requirement to use native Office Excel features to do calculations, for example.
  • The packaging API is a more lightweight approach, compared to the Web service APIs of Office Excel Services.
  • The structure of the workbook is such that we would have to make multiple calls to Office Excel Services to retrieve the data.

• Underwriter. For the underwriter approval flow, we are using Office SharePoint Server workflow–based tasks instead of creating custom e-mail messages. The main reason for doing this was to make sure the approval flow is integrated with the master loan workflow. Also, using the Office SharePoint Server tasks meant that we could trigger the workflow with the approval event without having to write any custom code.

• Excluded WPF, because this application was to be exposed as a thin extranet application for brokers. This would require a Windows Presentation Foundation Everywhere (WPF/E) download and installation for each client. WPF/E is the Microsoft solution for delivering rich, cross-platform, interactive experiences including animation, graphics, audio, and video for the Web and beyond using a subset of XAML (eXtensible Application Markup Language)–based Windows Presentation Foundation technology.
Developing from the Logical Architecture

Now that we understand the business architecture, we will describe the technical architecture in each logical tier of the solution. This is shown in Figure 1.

![Logical Architecture Diagram]

**Figure 1. Logical architecture for the OR-LOS solution**

Developing on a composite application layer as shown in Figure 1 provided us with the following advantages:

- Each component could be built and tested independently. This speeds development effort and ensures faster product releases.
- The data flow between components is well-defined. This helps to identify security requirements both at the message and transport levels.
- It was possible to plug in business services to them—and, if necessary, unplug them from—the platform.
- It was easier to perform unit testing. For example, during the development of Web services, we created a simple console application that could test the Web service. To address the need for sending valid inputs to the Web services, we created sample XML files. These files were read by the console application and the data sent to the Web service.
**Presentation Layer**

This layer serves as the user interface. We built this using Web parts hosted on Office SharePoint Server. Office SharePoint Service will provide the underpinnings for the application. There will be several services that can be inherited from this environment—specifically, the portal architecture that will be required to deploy Web parts for this solution.

**Application Services Layer**

This is a reusable layer in the architecture that will allow applications to use functionality such as Digital Rights Management, Document Libraries, Workflow, and so forth.

**Services Layer**

This layer provides an infrastructure to communicate messages. The Integration layer will use BizTalk.

**Business Rules Layer**

This layer provides centralized business rules to build consistency, reliability, efficiency, and cost reduction of system architectures. BizTalk Server has a built-in BRE that we will use.

**Orchestration Layer**

Process development and management occurs in this layer. BizTalk Server also has a robust orchestration engine for Business Activity Monitoring (BAM) types of activities. The architecture uses Business Process Execution Language (BPEL) extensively throughout to ensure interoperability between other orchestration systems.

**Data Services Layer**

Relational database services and management occur in this layer.

**Presentation Layer**

Here, we take each element of the user interface and discuss the implementation details, technologies and tools used, challenges, and so forth.

**The Lender Portal Built on Office SharePoint Server**

This section describes key areas of the lender's Web site that brokers and internal underwriters and processors will access. Loans will be registered, priced, locked, and underwritten through this Office SharePoint Server portal.

The lender portal is the main user interface in the OR-LOS solution. This Web site is set up as an Office SharePoint Server site collection. The site provides the entry point for all user actions, such as registration, product selection, and so on.
As shown in Figure 2, the Web site home page is created by composing a variety of Web parts. Some of the Web parts are out-of-the-box (OOB), such as the Business Data Catalog (BDC) Web part, whereas others are custom-built. Custom Web parts are required when the existing Web parts cannot fulfill all the requirements. To create custom Web parts:

1. Create a class and extend it from `System.Web.UI.WebControls.WebParts.WebPart`. Implement the code for the Web part functionality within the class.
2. Build the class into a Web part assembly.
3. Assign a strong name to the assembly.
4. Install the Assembly into the global assembly cache.
5. Add entry for `SafeControl` in the Web.config for the Office SharePoint Server site, as shown here.

```xml
<SafeControl Assembly="LoanApplicationWebPartLibrary, Version=1.0.0.0, Culture=neutral, PublicKeyToken=7b10cf3374fc1016" Namespace="LoanApplicationWebPartLibrary" TypeName="*" Safe="True" />
```
6. Add the Web part to the Web part gallery, so that the part can be added to pages.

Users can add private Web parts on the pages, as well as configure the properties of the parts. The parts can be modified using SharePoint Designer or by using the Office SharePoint Server browser–based interface. To add, delete, or modify Web parts, the users need to have Contribute user rights on the site. Also, any changes made to the pages need to be approved by users having Approval permissions.

The Web site itself is created using the SharePoint Central Admin Console, as shown in Figure 3.
As soon as the site is created, it is available for adding content like creating form libraries or adding custom Web parts.

Now that we described the various architecture elements and framework, we can dive into the various functional areas of the presentation layer.

**Loan Pipeline**

The loan pipeline provides a view of the current loans in the system. Brokers and underwriters alike will use the pipeline to view the status of customer loans.

The pipeline supports searching based on loan number and user name. Also, it restricts information, so that a broker is able only to view loans that have been created by the broker. The loan pipeline uses a BDC list Web part to fetch data from the database. By using BDC, we get features such as search, filtering, and paging out-of-the-box. The process for surfacing line-of-business (LOB) data using the Web part is to:

1. Create a BDC definition XML file. This file contains the definition of the entities that are mapped to LOB objects.
2. Upload the BDC xml into the SharePoint Central Admin Console. Office SharePoint Server validates the XML as it is loaded.
3. Add a BDC list Web part to the Web page and specify the name of the BDC entity.
The BDC definition is created as an XML file based on the metadata schema. For the OR-LOS, LoanOrigination is the top-level LOB system. Under Loan Origination, the metadata contains LoanInstance as the only business entity. The entity maps to a single row in the Loan table. This entity has the following key components:

1. Methods
   - LoanInstanceEnumerator. This is used by the BDC Search Web part to enumerate the entities.
   - LoanInstanceFinder. This is used by the BDC list Web part to list the instances. It uses filters for the loan number and last name of the borrower.

2. Actions
   - Open loan. This is used to open an Office InfoPath form with the loan details.

```xml
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<!-- loan metadata for Office 12 -->
<LobSystem
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://schemas.microsoft.com/office/2006/03/..."
```
Developing the Reference Application Pack for the Loan-Origination Systems (OR-LOS)

```xml
<BusinessDataCatalog BDCMetadata.xsd>
  <Type>Database</Type>
  <Version>1.0.0.9</Version>
  <Name>LoanOrigination</Name>
  <xmlns>http://schemas.microsoft.com/office/2006/03/BusinessDataCatalog>
    <Properties>
      <Property Name="WildcardCharacter" Type="System.String">%</Property>
    </Properties>
    <LobSystemInstances>
      <LobSystemInstance Name="LoanOrigination">
        <Properties>
          <Property Name="RdbConnection Data Source" Type="System.String">localhost</Property>
          <Property Name="RdbConnection Initial Catalog" Type="System.String">LoanOriginationDB</Property>
          <Property Name="RdbConnection Integrated Security" Type="System.String">SSPI</Property>
        </Properties>
      </LobSystemInstance>
    </LobSystemInstances>
    <Entities>
      <Entity EstimatedInstanceCount="1000" Name="LoanInstance">
        <Properties>
          <Property Name="Title" Type="System.String">Number</Property>
          <Property Name="Audit" Type="System.Boolean">true</Property>
        </Properties>
        <Identifiers>
          <Identifier Name="LoanID" />
        </Identifiers>
      </Entity>
    </Entities>
  </BusinessDataCatalog>
```
TypeName="System.String" />
      </MethodInstances>
    </Methods>
  </Identifiers>
</TypeDescriptors>
</TypeDescriptor>
</TypeDescriptors>
</TypeDescriptor>
</Parameters>
</MethodInstances>
<Method Instances>
  <MethodInstance
    Name="LoanInstanceEnumeratorInstance" Type="IdEnumerator"
    ReturnParameterName="LoanInstances"/>
</MethodInstances>
</TypeDescriptors>
</TypeDescriptor>
</TypeDescriptors>
</TypeDescriptor>
</Parameters>
</MethodInstances>
<Method Instances>
  <MethodInstance
    Name="LoanInstanceEnumeratorInstance" Type="IdEnumerator"
    ReturnParameterName="LoanInstances"/>
</MethodInstances>
</TypeDescriptors>
</TypeDescriptor>
</TypeDescriptors>
</TypeDescriptor>
</Parameters>
</MethodInstances>
<Method Instances>
  <MethodInstance
    Name="LoanInstanceEnumeratorInstance" Type="IdEnumerator"
    ReturnParameterName="LoanInstances"/>
</MethodInstances>
</TypeDescriptors>
</TypeDescriptor>
</TypeDescriptors>
</TypeDescriptor>
</Parameters>
</MethodInstances>
<Method Instances>
  <MethodInstance
    Name="LoanInstanceEnumeratorInstance" Type="IdEnumerator"
    ReturnParameterName="LoanInstances"/>
</MethodInstances>
</TypeDescriptors>
</TypeDescriptor>
</TypeDescriptors>
</TypeDescriptor>
</Parameters>
</MethodInstances>
<Method Instances>
  <MethodInstance
    Name="LoanInstanceEnumeratorInstance" Type="IdEnumerator"
    ReturnParameterName="LoanInstances"/>
</MethodInstances>
</TypeDescriptors>
</TypeDescriptor>
</TypeDescriptors>
</TypeDescriptor>
</Parameters>
</MethodInstances>
<Method Instances>
  <MethodInstance
    Name="LoanInstanceEnumeratorInstance" Type="IdEnumerator"
    ReturnParameterName="LoanInstances"/>
</MethodInstances>
</TypeDescriptors>
</TypeDescriptor>
</TypeDescriptors>
</TypeDescriptor>
</Parameters>
</MethodInstances>
<Method Instances>
  <MethodInstance
    Name="LoanInstanceEnumeratorInstance" Type="IdEnumerator"
    ReturnParameterName="LoanInstances"/>
</MethodInstances>
</TypeDescriptors>
</TypeDescriptor>
</TypeDescriptors>
</TypeDescriptor>
</Parameters>
</MethodInstances>
<Method Instances>
  <MethodInstance
    Name="LoanInstanceEnumeratorInstance" Type="IdEnumerator"
    ReturnParameterName="LoanInstances"/>
</MethodInstances>
</TypeDescriptors>
</TypeDescriptor>
</TypeDescriptors>
</TypeDescriptor>
</Parameters>
</MethodInstances>
<Method Instances>
  <MethodInstance
    Name="LoanInstanceEnumeratorInstance" Type="IdEnumerator"
    ReturnParameterName="LoanInstances"/>
</MethodInstances>
</TypeDescriptors>
</TypeDescriptor>
</TypeDescriptors>
</TypeDescriptor>
</Parameters>
</MethodInstances>
<Method Instances>
  <MethodInstance
    Name="LoanInstanceEnumeratorInstance" Type="IdEnumerator"
    ReturnParameterName="LoanInstances"/>
</MethodInstances>
</TypeDescriptors>
</TypeDescriptor>
</TypeDescriptors>
</TypeDescriptor>
</Parameters>
</MethodInstances>
<Method Instances>
  <MethodInstance
    Name="LoanInstanceEnumeratorInstance" Type="IdEnumerator"
    ReturnParameterName="LoanInstances"/>
</MethodInstances>
</TypeDescriptors>
</TypeDescriptor>
</TypeDescriptors>
</TypeDescriptor>
</Parameters>
</MethodInstances>
<Method Instances>
  <MethodInstance
    Name="LoanInstanceEnumeratorInstance" Type="IdEnumerator"
    ReturnParameterName="LoanInstances"/>
</MethodInstances>
</TypeDescriptors>
</TypeDescriptor>
</TypeDescriptors>
</TypeDescriptor>
</Parameters>
</MethodInstances>
<Method Instances>
  <MethodInstance
    Name="LoanInstanceEnumeratorInstance" Type="IdEnumerator"
    ReturnParameterName="LoanInstances"/>
</MethodInstances>
</TypeDescriptors>
</TypeDescriptor>
</TypeDescriptors>
</TypeDescriptor>
</Parameters>
</MethodInstances>
<Method Instances>
  <MethodInstance
    Name="LoanInstanceEnumeratorInstance" Type="IdEnumerator"
    ReturnParameterName="LoanInstances"/>
</MethodInstances>
</TypeDescriptors>
</TypeDescriptor>
</TypeDescriptors>
</TypeDescriptor>
</Parameters>
</MethodInstances>
<Method Instances>
  <MethodInstance
    Name="LoanInstanceEnumeratorInstance" Type="IdEnumerator"
    ReturnParameterName="LoanInstances"/>
</MethodInstances>
</TypeDescriptors>
</TypeDescriptor>
</TypeDescriptors>
</TypeDescriptor>
</Parameters>
</MethodInstances>
<Method Instances>
  <MethodInstance
    Name="LoanInstanceEnumeratorInstance" Type="IdEnumerator"
    ReturnParameterName="LoanInstances"/>
</MethodInstances>
</TypeDescriptors>
</TypeDescriptor>
</TypeDescriptors>
</TypeDescriptor>
</Parameters>
</MethodInstances>
<Method Instances>
  <MethodInstance
    Name="LoanInstanceEnumeratorInstance" Type="IdEnumerator"
    ReturnParameterName="LoanInstances"/>
</MethodInstances>
</TypeDescriptors>
</TypeDescriptor>
</TypeDescriptors>
</TypeDescriptor>
</Parameters>
</MethodInstances>
<Method Instances>
  <MethodInstance
    Name="LoanInstanceEnumeratorInstance" Type="IdEnumerator"
    ReturnParameterName="LoanInstances"/>
</MethodInstances>
</TypeDescriptors>
</TypeDescriptor>
</TypeDescriptors>
</TypeDescriptor>
</Parameters>
</MethodInstances>
<Method Instances>
  <MethodInstance
    Name="LoanInstanceEnumeratorInstance" Type="IdEnumerator"
    ReturnParameterName="LoanInstances"/>
</MethodInstances>
<Method Name="LoanInstanceFinder">
  <Properties>
    <Property Name="RdbCommandText" Type="System.String">
      GetLoans
    </Property>
    <Property Name="RdbCommandType" Type="System.String">StoredProcedure</Property>
  </Properties>
  <FilterDescriptors>
    <FilterDescriptor Type="Wildcard" Name="LoanID">
      <Properties>
        <Property Name="UsedForDisambiguation" Type="System.Boolean">true</Property>
      </Properties>
    </FilterDescriptor>
    <FilterDescriptor Name="UserInformation">
      <Properties>
        <Property Name="UsedForDisambiguation" Type="System.Boolean">false</Property>
      </Properties>
    </FilterDescriptor>
    <FilterDescriptor Type="Wildcard" Name="LastName">
      <Properties>
        <Property Name="UsedForDisambiguation" Type="System.Boolean">true</Property>
      </Properties>
    </FilterDescriptor>
  </FilterDescriptors>
  <Parameters>
<Parameter Direction="In"
Name="&loanID">
   <TypeDescriptor Name="LoanID"
Type="System.String"
IdentifierName="LoanID">
      <DefaultValues>
         <DefaultValue
           MethodInstanceName="LoanInstanceFinderInstance"
           Type="System.String">%</DefaultValue>
         <DefaultValue
           MethodInstanceName="LoanInstanceSpecificFinderInstance"
           Type="System.String">%</DefaultValue>
      </DefaultValues>
   </TypeDescriptor>
</Parameter>

(Parameter Direction="In"
Name="&UserName">
   <TypeDescriptor
Name="UserName" TypeName="System.String"
AssociatedFilter="UserInformation"/>
</Parameter>

(Parameter Direction="In"
Name="&name">
   <TypeDescriptor Name="Name"
Type="System.String" AssociatedFilter="LastName">
      <DefaultValues>
         <DefaultValue
           MethodInstanceName="LoanInstanceFinderInstance"
           Type="System.String">%</DefaultValue>
         <DefaultValue
           MethodInstanceName="LoanInstanceSpecificFinderInstance"
           Type="System.String">%</DefaultValue>
      </DefaultValues>
   </TypeDescriptor>
</Parameter>

<Parameter Name="LoanInstances"
Direction="Return" >
<TypeDescriptor Name="LoanInstances" TypeName="System.Data.IDataReader, System.Data, Version=2.0.3600.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" IsCollection="true">
  <TypeDescriptors>
    <TypeDescriptor Name="LoanInstance" TypeName="System.Data.IDataRecord, System.Data, Version=2.0.3600.0, Culture=neutral, PublicKeyToken=b77a5c561934e089">
      <TypeDescriptors>
        <TypeDescriptor Name="Number" TypeName="System.String">
          <LocalizedDisplayNames>
            <LocalizedDisplayName LCID="1033">Number</LocalizedDisplayName>
          </LocalizedDisplayNames>
          <Properties>
            <Property Name="DisplayByDefault" Type="System.Boolean">true</Property>
          </Properties>
        </TypeDescriptor>
        <TypeDescriptor Name="Name" TypeName="System.String">
          <LocalizedDisplayNames>
            <LocalizedDisplayName LCID="1033">Primary Borrower</LocalizedDisplayName>
          </LocalizedDisplayNames>
          <Properties>
            <Property Name="DisplayByDefault" Type="System.Boolean">true</Property>
          </Properties>
        </TypeDescriptor>
        <TypeDescriptor Name="Amount" TypeName="System.String">
          <LocalizedDisplayNames>
            <LocalizedDisplayName LCID="1033">Amount</LocalizedDisplayName>
          </LocalizedDisplayNames>
          <Properties>
            <Property Name="DisplayByDefault" Type="System.Boolean">true</Property>
          </Properties>
        </TypeDescriptor>
      </TypeDescriptors>
    </TypeDescriptor>
  </TypeDescriptors>
</TypeDescriptor>
<Properties>
  <Property Name="DisplayByDefault" Type="System.Boolean">true</Property>
</Properties>

<TypeDescriptor Name="Status" TypeName="System.String">
  <LocalizedDisplayNames>
    <LocalizedDisplayName LCID="1033">Status</LocalizedDisplayName>
  </LocalizedDisplayNames>
  <Properties>
    <Property Name="DisplayByDefault" Type="System.Boolean">true</Property>
  </Properties>
</TypeDescriptor>

<TypeDescriptor Name="ProductName" TypeName="System.String">
  <LocalizedDisplayNames>
    <LocalizedDisplayName LCID="1033">Product Name</LocalizedDisplayName>
  </LocalizedDisplayNames>
  <Properties>
    <Property Name="DisplayByDefault" Type="System.Boolean">true</Property>
  </Properties>
</TypeDescriptor>

<TypeDescriptor Name="ExpiryDate" TypeName="System.String">
  <LocalizedDisplayNames>
    <LocalizedDisplayName LCID="1033">Expiry Date</LocalizedDisplayName>
  </LocalizedDisplayNames>
  <Properties>
    <Property Name="DisplayByDefault" Type="System.Boolean">true</Property>
  </Properties>
</TypeDescriptor>
Developing the Reference Application Pack for the Loan-Origination Systems (OR-LOS) 20

<TypeDescriptor Name="CreditFilePath"
    TypeName="System.String">
    <LocalizedDisplayNames>
        <LocalizedDisplayName LCID="1033">Credit File</LocalizedDisplayName>
    </LocalizedDisplayNames>
    <Properties>
        <Property Name="DisplayByDefault" Type="System.Boolean">true</Property>
    </Properties>
</TypeDescriptor>

<TypeDescriptor Name="ModifiedDate"
    TypeName="System.String">
    <LocalizedDisplayNames>
        <LocalizedDisplayName LCID="1033">Modified Date</LocalizedDisplayName>
    </LocalizedDisplayNames>
    <Properties>
        <Property Name="DisplayByDefault" Type="System.Boolean">true</Property>
    </Properties>
</TypeDescriptor>

<MethodInstances>
    <MethodInstance Name="LoanInstanceFinderInstance" Type="Finder"
        ReturnParameterName="LoanInstances" />
    <MethodInstance Name="LoanInstanceSpecificFinderInstance" Type="SpecificFinder"
        ReturnParameterName="LoanInstances" />
</MethodInstances>
Loan-Registration Forms

The loan registration form is used by the broker to save details of the user in the system. It is also used to lock and float a loan rate.

The loan registration form is created using a combination of Office InfoPath 2007 with Form Services. Form Services allows Office InfoPath forms to be viewed as a Web part through a portal environment. The forms are designed in the Office InfoPath form designer. The data source for the forms is based on MISMO XML Schema Definition (XSD) schema.
Figure 5. Data source for the loan-registration form

Because the MISMO standard is available only as document-type definition (DTD) files, we needed to convert the DTD files into XSD schemas. Visual Studio 2005 provides an interface for doing this conversion, as shown in Figure 6.
Figure 6. Converting DTDs to XSDs in Visual Studio

As soon as the form template is associated with the XSD schema, the schema file is stored within the form template. So, to make any changes to the form schema, we need to update the stored schema.

To alter a schema after the Office InfoPath template (.xsn) has been created:

1. Rename the .xsn file template to .cab.
2. Extract the files from the cab into a folder.
3. Update the schema files located in the folder.
4. Repackage the files into a cab file. For this, you can use the Windows SDK tool "cabarc."
5. Rename the .cab to .xsn file.

Another requirement for the loan registration process was to provide a sequence of pages to the user. To achieve this, we use the View feature of Office InfoPath. Views in Office InfoPath provide a sequence of screens. Each screen is mapped to a fragment of the underlying form XML. The sequence can be configured by associating actions on form buttons.

We used the data validation for the controls feature of Office InfoPath to restrict user input. For example, for the SSN field, we configured the form in such a way that the text box will accept Social Security numbers only in the valid format. The configuration can be done as shown in Figure 7.
Figure 7. Form validation in Office InfoPath

At run time, the screen appears as shown in Figure 8.
Figure 8. The user interface, showing restricted input format

XmlFormView Web Part

The loan-application form needed to be displayed in the Office SharePoint Server site in the context of the Web page. This posed a challenge while using Form Services, because the browser renders the form in its own page. To solve this issue, we had to use the XmlFormView Web part. This Web part is given by ASP.NET 2.0. The assembly to be included in the references is Microsoft.Office.InfoPath.Server.dll. The Web part can render Office InfoPath form XML within Web pages. So, that solved the first problem for us. However, we needed to pass custom parameters to the Web part through the name of the document library where the form template is located. (Note that the XmlFormView Web part contains properties that store the form location.) However, we needed to surface this property all the way up to the Office SharePoint Server page. To solve this, we created a custom Web part that wraps the XmlFormView Web part. By making the form location customizable, we have created a generic Office InfoPath Web part that can be placed on any Web page and hooked up to the required form template.

Figure 9 shows how the Web part is configured.
Figure 9. Configuration parameters for loan-application Web part

Products and Pricing Flow

One of the flows in the Office InfoPath form design required that we make a call to the Products Web service to fetch qualifying products. For this, we needed to add code-behind using Visual Studio for Architects 2005. The Office InfoPath Designer provides an interface for creating the Visual Studio project, as shown in Figure 10.
The following code is executed by the code-behind to get the rates.

```csharp
using (StreamReader reader = new StreamReader(xsltPath))
{
    xsltData = reader.ReadToEnd();
}
using (StringReader stringReader = new StringReader(xsltData))
{
    using (XmlReader reader = XmlReader.Create(stringReader))
    {
        outputData = Util.TransformXmlString(inputData, reader);
    }
}
// create input parameter for Web service
LOAN_APPLICATION loan = CreateLoan(outputData);
// call Web service
```
When the form is published, it automatically builds the underlying project. Because the Office InfoPath form contains code-behind, we have to take some additional steps in making it available to fill out. These steps are collectively called "admin deployment":

1. The form template needs to be published to a location that is accessible to an administrator.
2. The administrator needs to upload the form template to the form-template collection. The SharePoint Central Admin provides a screen for performing the form-template upload.
3. The form template needs to be activated to a particular site collection.

One of the requirements of the product and pricing flows is the concept of **blackout**. This is the period when the daily rates are not published yet. During blackout, borrowers cannot lock on any rates. We needed a special form to handle this condition and display the correct error messages. To implement this, we have added a flag in the configuration file for the Web service that returns the daily rates. Based on whether the flag is **true** or **false**, the Web service returns either the product data or an error XML. The form handles the return values and displays the correct content. This can be seen from the following code examples.
### Web Service Code

```csharp
bool blackout = IsBlackoutOn();
string returnValue = string.Empty;
if (!blackout)
{
    // BizTalk invocation
    object[] invokeResults = this.Invoke("GetProducts", invokeParams,
inParamInfos, outParamInfos, 0,
bodyTypeAssemblyQualifiedName, inHeaders, inoutHeaders, out inoutHeaderResponses, out
outHeaderResponses, null, null, null, out
unknownHeaderResponses, false, false);
    returnValue = ((string)(invokeResults[0]));
}
else
{
    returnValue = "<Output
xmlns="http://BizTalkSchemas.APP_MORTGAGE_APPLICATION_v2.1.LOAN_APPLICATION">
<Blackout>true</Blackout></Output>";
}
```

### Form Code

```csharp
// check if blackout is on
bool isBlackout = CheckForBlackout(productsNav, manager);

if (!isBlackout)
{
    // display products
}
else
{
    message = "Products cannot be viewed during blackout. Please try again
later.";
}
```
Underwriter Approval Form

This form is used by the underwriter to approve the loan. It displays the loan details along with a text area where the user can enter comments.

This form is associated with a task in the master loan workflow. The form design is based on a sample workflow task form provided by Office SharePoint Server product team. Because the loan is supposed to display loan-summary data, we have a secondary data source in the form. The data source is connected to a Web service. The Web service is used to retrieve loan summary. The following code example shows how the Web service is invoked and how the return value used to populate the form.

```csharp
public void FormEvents_Loading(object sender, LoadingEventArgs e)
{
    // get the value of the loan number
    DataConnectionCollection dataConnections = this.DataConnections;
    XPathNavigator docNav = MainDataSource.CreateNavigator();
    XPathNavigator node = docNav.SelectSingleNode("/my:myFields/my:Instruction", this.NamespaceManager);
    string loanNumber = node.Value;
    // set the value of the loan number in the Web service retrieval field
    DataSource targetDataSource = DataSources["GetLoanData"];
    XPathNavigator targetNode = targetDataSource.CreateNavigator();
    XPathNavigator itemIdNode =
        targetNode.SelectSingleNode
        ("/dfs:myFields/dfs:queryFields/tns:GetLoanData/tns:loanNumber", this.NamespaceManager);
    itemIdNode.SetValue(loanNumber);
    // execute the Web service to get the data.
    targetDataSource.QueryConnection.Execute();
    // get the file data returned by the Web service call
    XPathNavigator creditFileNode =
        targetNode.SelectSingleNode
        ("/dfs:myFields/dfs:dataFields/tns:GetLoanDataResponse/tns:GetLoanDataResult" + "/tns:creditInfo/tns:fileContents", this.NamespaceManager);
    string fileContents = creditFileNode.Value;
    // get the file name returned by the Web service call
    XPathNavigator creditFileNameNode =
        targetNode.SelectSingleNode
        ("/dfs:myFields/dfs:queryFields/tns:GetLoanDataResponse/tns:GetLoanDataResult/tns:creditInfo/tns:fileName", this.NamespaceManager);
    string fileName = creditFileNameNode.Value;
}
```
The form design appears as shown in Figure 12.
Figure 12. Loan-approval form template, design view

The approval form template is installed in the Office SharePoint Server farm as part of the workflow installation. This can be seen from the form-template identifier in the workflow XML file shown in Figure 13.
Figure 13. Task-form association in the workflow definition

**Daily Rates Workbook**

The daily rates Office Excel file contains the values for the loan products, such as adjustable-rate mortgages (ARMs) and fixed-rate products.
The values in the Office Excel file are modified by secondary market processors. The file itself is stored in Office Excel Services on the server. This allows us to publish Web-based views of the file to restrict user actions on the file.

The main requirement here was to read the data from the Office Excel file and then store it both into a database and into an XML file. For reading the data, we are using the `System.IO.Packaging` APIs available in .NET 3.0. To make data retrieval easier, we added named ranges for the data. Then, the code uses the OpenXml file format to read specific sheets from the Office Excel file.

**Application Services Layer**

This section describes the reusable services that our application will use. These services include workflow that exposes the lending business processes; security features, such as IRM; a portal for displaying and communicating our loan data; and storage mechanisms to retain information centrally.
Master Loan Workflow

The master loan flow (MLF) is the backbone of the lending process. The MLF controls of the varying aspects of the loan process. MLF manages both WF workflows and in BizTalk Server 2006 orchestrations. There are several advantages to doing this. One advantage is that the process can be managed much more effectively through a holistic view. Second, long-running workflow is abstracted out as a service to which other processes can attach. This enables the solution to interoperate much more flexibly with other lending solutions.

The MLF uses the WF state-machine style of workflow.

We choose to use the state machine, instead of the sequential flow, for the following reasons:

- The loan transitions through multiple states in its lifetime.
- The loan has to respond to asynchronous events during its lifetime.

For the interaction between documents and events to occur, the MLF is associated with a form library in Office SharePoint Server. This library hosts the template of the Office InfoPath loan-application form that was referenced in Figure 11.
Figure 16. Master loan workflow state machine

The state machine in Figure 16 maps directly back to the business process we want to expose for loan processing.

Each of the states in the workflow contains fault-handlers for handling errors occurring in state-code execution. To implement the expiration feature for a loan, we have used delay timers within the states. These timers are fired when a certain amount of time elapses. The loan workflow responds to events that are raised by Office SharePoint Server. These events are raised in response to user actions such as form creation, form updates, or form deletion. In response to these events, the workflow performs different activities based on the current state of the loan. Whenever events/activities are completed, the workflow logs the result in the form library. This logging is done using the LogToHistoryActivity given by the Office SharePoint Server Workflow activity extensions. This result can be viewed by navigating to the form in the form library. Whenever the workflow instance is waiting for any event to occur, Office SharePoint Server dehydrates the workflow instance until the event is raised.

The workflow is created using the Office SharePoint Server workflow extensions available for Visual Studio 2005. The extensions add the relevant scripts required for installing the workflows onto an Office SharePoint Server farm. The developer just needs to replace the values in the designated placeholders and execute the script.
To deploy the workflow, take the following steps:

1. Deploy the workflow as a feature in the farm.
2. Activate the feature to the site collection.
3. Associate the workflow with the loan form library.

**Products and Pricing Workflow**

The products and pricing workflow is used for publishing daily rates.

![Diagram of Products and Pricing Workflow](image-url)

**Figure 17. Products and pricing workflow overview**

As shown in Figure 17, the workflow is associated with the document library that contains the workbook. The workflow kicks in when the processor updates the rates in the rate workbook and saves the document. The workflow also receives the information about which item in the library has been modified. Using this information, the workflow reads the data from the Office Excel file. It then executes the following actions:

1. It sends alerts to concerned personnel that the new rates are available.
2. It sends the document over for approval.
3. It saves the data from the file into the database and the XML files.
4. It updates the Blackout flag to indicate the Blackout is set to an "Off" status.

The workflow is a Windows Workflow Framework–based sequential workflow. We decided to go for a sequential workflow in this case, because the process is straightforward. Also, no asynchronous events need to be handled during the workflow execution.
Figure 18. Product and pricing sequential workflow

The workflow is also created within the Visual Studio development environment. It is then deployed to Office SharePoint Server using the standard deployment scripts. The workflow uses the OpenXmlPackage custom class to read the data from the Office Excel file. The following code example shows how the APIs are invoked.

```csharp
private Hashtable GetRatesFromWorkbook()
{
    Hashtable data = null;
    using (SPSite thisSite = new SPSite(workflowProperties.SiteId))
    {
        using (SPWeb myWeb = thisSite.OpenWeb(workflowProperties.WebId))
        {
            SPList myList = myWeb.Lists[workflowProperties.ListId];
            SPLISTItem myItem = myList.GetItemById(workflowProperties.ItemId);
            SPFile file = myItem.File;
            currentUser = file.ModifiedBy.LoginName;
        }
    }

    return data;
}
```
string fileUrl = thisSite.MakeFullUrl(file.Url);
Util.Log(Constants.LogFileName, "fileUrl:" + fileUrl);

// get the named ranges that contain the values
Hashtable namedRanges = GetNamedRanges();

// the hashtable contains a table for each product listed above
// each table has N rows. Each row has 2 cells, for rate and discount respectively
data = OpenXmlUtil.GetData(namedRanges, file.OpenBinaryStream(),
   "http://schemas.openxmlformats.org/spreadsheetml/2006/7/main",
   sheetName);
}
return data;

---

**Lending Message Bus**

The OR-LOS solution requires interaction with third-party services. These services provide loan-related data for underwriters to make decisions. Such data would include retrieving flood information or insurance and pulling credit from the credit bureaus for the user. Because these services are provided by external systems, we are using BizTalk Server to integrate the loan workflow with the services. In this way, BizTalk Server provides the enterprise service bus for OR-LOS. This integration is implemented using the orchestrations discussed in the later sections.

The orchestrations are created and deployed to the host using Visual Studio. As soon as it is deployed, we use the BizTalk Server publishing tool to publish the orchestration as a Web service. The publishing tool creates the virtual directory under the default root within Microsoft Internet Information Services (IIS). This virtual directory hosts the Web service that will be used to activate the orchestration. The Web service contains all the code for sending the requests to the correct orchestration.
**Credit Orchestration**

This orchestration is used to retrieve credit information for an applicant.

![Credit orchestration design](image)

**Figure 19. Credit-orchestration design**

The orchestration accepts a MISMO credit request as input. It then calls a stub Web service to fetch credit data. The orchestration maps the data into a request type compatible with the Web service. The stub Web service fetches data from a database and sends back a credit response. The stub Web service is created using the Visual Studio ASMX Web Service project template. This creates a virtual directory under the default root in IIS. As soon as the virtual directory is created, Web message types need to be created within the BizTalk Server project. This can be done by using the Add Web Reference Wizard in the project. This wizard requires the URL of the Web service. After Web message types have been added, send ports in the orchestration can be configured to send the correct message type. Also, the return type of the Web service needs to be set inside the receive port for each of the send port. As soon as the Web service sends data, the orchestration converts the credit response into a MISMO credit response and sends it back to the calling process.
Figure 20. Web service for the Credit orchestration
Flood Orchestration

This orchestration is used to retrieve Flood information for a property.

Figure 21. Flood-orchestration design view

This orchestration is similar to the earlier Credit orchestration. One key difference is that we needed to build this to show asynchronous communication. In the case of Flood data, the response is not obtained immediately. To demonstrate this, we decided to make use of the FileAdapters within BizTalk Server. The orchestration copies the MISMO flood request as a file in a local folder. The orchestration watches for response files in another folder. As soon as a response file is available, it is picked up by BizTalk Server and processed. If the response field is not in the correct format, it is rejected, and the orchestration goes back to watching for new files. The orchestration converts the credit response into a MISMO credit response and sends it back. The orchestration is finally published as an IIS ASMX Web service.
Figure 22. Web service for the flood orchestration
Orchestration and BRE

The Products orchestration is used during the registration process to retrieve the list of eligible products. Here, we discuss the details of the orchestration, including how it uses the BizTalk Server Business Rules Engine (BRE) to filter the rates.

![Orchestration and BRE diagram](image)

**Figure 23. Products-orchestration design view**

The orchestration receives the loan details from the master loan workflow. It then reads products and rates from the cache. The cache is implemented using the Microsoft Caching Application Block, which is part of the Enterprise Library. The application block can be used without modification and provides all the functionality needed to retrieve, add, and remove cached data. Configurable expiration and scavenging policies are also part of the application block’s functionality. We needed to install the application block assembly in the global assembly cache to make it accessible from within BizTalk Server orchestration.

The products, rates, and user data are sent to the BRE.
Figure 24. Connecting products orchestration with BRE

The BRE returns the list of qualifying rates and their products. By using rules stored in the BRE, we are separating the rules from the client code. This allows a business analyst to modify the rules without recompiling the client code.
The orchestration copies the selected products and rates and sends them back as XML. The orchestration is developed using BizTalk Server extensions for Visual Studio 2005. The extensions enable deployment of the orchestrations through the Visual Studio environment. As soon as the orchestration is deployed, the user needs to perform the following steps in the BizTalk Administration Console before the orchestration can be used:

1. Configure the Send Ports for the orchestration.
2. Configure the Receive Ports for the orchestration.
3. Configure the Receive Locations for the orchestration.
4. Check if there are suspended instances of the same orchestration. If yes, those instances need to be removed.
5. Restart the host application that hosts the orchestration.
6. Check to see if there are any errors while starting each of the orchestrations.
Figure 26. Orchestration status in the BizTalk Administration Console
Finally, the orchestration is published as an IIS ASMX Web service.

![Image of Web service interface](image)

Figure 27. Web service for products orchestration

**Data Services Layer**

All data access within the OR-LOS is provided by the Microsoft Data Access Application block, which is a part of the Enterprise Library. The application block provides an abstract layer for interacting with a data source. For example, it provides the `SqlDatabase` class that contains APIs such as `ExecuteNonQuery()`. Also, it takes care of managing critical resources such as database connections. To use the data application block, one needs to include the relevant assemblies in the project, declare the namespaces in the class file, and then invoke the relevant APIs from the client code. If the client code resides within the global assembly cache, the application block assemblies also need to be installed in the global assembly cache.

The following code example shows how to use the Data Access Application Block to fire an update query.

```csharp
private void SaveCreditDataActivity_ExecuteCode(object sender, EventArgs e)
{
    SqlDatabase sqlDatabase = new SqlDatabase(this.ConnectionString);
    using (DbCommand dbCommand = sqlDatabase.GetStoredProcCommand("WriteCreditData"))
```
Reporting Services

We use Reporting Services for creating reports that are used by the underwriter to view the loan pipeline.

SQL Server Reporting Services provide Web parts that can be used with Office SharePoint Server 2007. These Web parts need to be installed on an Office SharePoint Server farm before they can be used on a page. The installation is done by using the `stsadm -addwppack` command. The two parameters required for the command are a path to the Web part package that is installed by Reporting Services, and a URL to the site where the Web parts need to be installed. This command performs the following actions:

1. It installs the Web part assembly in the global assembly cache.
2. It adds the following entry in the Web.config for the site.

```xml
<SafeControl Assembly="RSWebParts, Version=8.0.242.0, Culture=neutral, PublicKeyToken=89845dcd8080cc91" Namespace="Microsoft.ReportingServices.SharePoint.UI.WebParts" TypeName="*" Safe="True" />
```
As soon as the Web parts are installed, to view reports from Office SharePoint Server, we need to add the Report Viewer Web parts to a page. Figure 28 shows how the Web part is configured.

Figure 28. Configuring the Report Viewer Web part

The report in the OR-LOS solution is created by using the Visual Studio SQL Reporting Services designer. This add-on allows the developer to perform all the functions required to create reports. The process involves the following:

1. Create a Reporting Services project in Visual Studio.
2. Define data sources for the reports. This is done by adding database servers and databases to the project.
3. Add reports to the project. Add the data-retrieval mechanism using stored procedures or SQL queries. You can also configure the look and feel of the report through the designer tool.
4. As soon as the report design is complete, it can be deployed to the Reporting Services server.
Database Diagram

Next, we discuss the design of the database used for the OR-LOS solution.

Figure 29 shows the data model for the OR-LOS solution. The model was developed using the following inputs:

- MISMO XML schemas. Because we are using the MISMO format for our messaging, we decided to use that for creating the base data model. This meant that we maintained consistency in the parameter names in the format and the database fields.

- Operational requirements. We added tables and fields during the course of development also. For each field, we evaluated whether it made good design sense to store the values in the database. For example, we have a table called "TaskMapping." This stores the mapping between workflow tasks, loan IDs, and task types (such as Flood or Approval). Similarly, we have added fields in the Loan Table for storing the status of the loan.

Table 1 shows the important tables used in the solution.

Table 1. List of database tables used in the OR-LOS solution

<table>
<thead>
<tr>
<th>Table name</th>
<th>Purpose</th>
<th>Tables referenced</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loan</td>
<td>Stores the loan details such as amount, purpose, term</td>
<td>Product, Property</td>
</tr>
<tr>
<td>Person</td>
<td>Stores personal details such as name, Social Security number</td>
<td>-</td>
</tr>
<tr>
<td>Borrower</td>
<td>Stores the details of the borrowers of each loan (for example, primary borrower)</td>
<td>Person, Loan</td>
</tr>
<tr>
<td>Property</td>
<td>Stores property details such as street, age</td>
<td>-</td>
</tr>
<tr>
<td>Product</td>
<td>Stores the loan products (for example, 35-Year Fixed)</td>
<td>-</td>
</tr>
<tr>
<td>Rate</td>
<td>Stores the different rate combinations (for example, value, conditions, points for each product)</td>
<td>Product</td>
</tr>
<tr>
<td>Credit</td>
<td>Stores the credit-file information related to a loan including, any files received from the credit agency</td>
<td>Loan</td>
</tr>
<tr>
<td>Flood</td>
<td>Stores the Flood information related to a property, including any files received from the Flood-information provider</td>
<td>Loan</td>
</tr>
</tbody>
</table>
Figure 29. Database structure for the OR-LOS solution

Stored Procedures

The OR-LOS uses stored procedures extensively to interact with the database. This involves both read and write operations. Using stored procedures ensures the following:

- The response time is improved, because the procedures are compiled on first use.
- Security is improved, because SQL injection attacks are avoided.
- Maintainability is improved, because the stored procedure provides a logical unit of code that can be tested separately.

This list shows the stored procedures used:

- **AddLoanComments.** Used to save approver or submitter comments to the database.
- **GetLoanApplication.** Used to retrieve the loan details from the database. This is used by the loan approval form to get the loan data.
- **GetLoanApplicationSummary.** Used to retrieve loan-summary details. This is used to create the content of the loan approval e-mail.
• **GetLoans.** This is used to retrieve the list of loans that meet some criteria.

• **GetTaskMapping.** Used to retrieve the task ID related to the Flood or Approval task.

• **WriteCreditData.** Used to save credit information to the database.

• **WriteFloodData.** Used to save flood data to the database.

• **WriteLoanApplication.** Used to save loan data into the database.

• **WriteNewRates.** Used by the Pricing workflow to save the daily rates into the database.

• **WriteOrUpdateTask.** Used to update the details of Flood or Approval task details.

**Limitations and Challenges**

The following features are not supported by Form Services:

• Master-child controls

• Dialog boxes

• IRM permissions cannot be applied to the task e-mail, because the task e-mail is managed by Office SharePoint Server.

• BDC provides read-only views to LOB data.

**Fitting It All Together**

Here, we take one of the business scenarios (Products and Pricing, Loan Registration, and Underwriting Wholesale Loans) and show how each of the design components interact together. Key points include the following:

**Registration Process**

Here, a new user enters data into the system. This data consists of:

• Personal information

• Property information

This data is saved into the database by the master loan workflow. As soon as the personal and property data is entered, the system makes a call to BizTalk to retrieve the eligible rates for the user. The user can then select rates for the loan and lock the loan. At the point in which the user locks a loan, the master loan workflow requests third-party services. After the responses to all the services are returned, the loan is sent to the underwriter for approval. As soon as the underwriter approves, the loan workflow is complete.
Figure 30. Mortgage-loan life cycle

WF to BizTalk

The master loan workflow talks to the BizTalk orchestration using the IIS Web services. The Web services act as a proxy and invoke the respective orchestrations. If any error occurs while executing the orchestration, the orchestration is suspended.

Office InfoPath Code-Behind to BizTalk WS

For retrieving the eligible products' data, the code-behind accesses the Product orchestration through a proxy Web service. The data returned by the Web service is in the form of XML. The code-behind converts this XML to use within the form and then updates the display.

Office InfoPath to WF

Office InfoPath and the master loan workflow do not interact directly in the application. The interaction happens through events raised whenever the Office InfoPath form is created or updated in the form library. The Create or Update events are recognized by Office SharePoint Server. Office SharePoint Server then finds the corresponding workflow instance and sends the event and its associated data. The workflow also receives the details about which item has been updated. Using this, it retrieves the form XML. Then, it processes the XML based upon the user action. For example, if the user has just saved a loan, the workflow just writes the data into the database. On the other hand, if the user has locked on a rate, in addition to saving the data into the database, the workflow also kicks off the process of retrieving third-party services.
Conclusion

With the right tools and processes, building enterprise solutions on Microsoft technologies is a viable option for financial-services institutions. We have shown developers how to build secure, highly reliable, and scalable solutions that meet their daily demands.

Using the Windows Workflow Foundation (WF) that is built into the .NET Framework 3.0, coupled with Office SharePoint Server, makes it easier for developers to build feature-rich solutions with out-of-the-box technologies—allowing developers to focus less on writing code for plumbing or infrastructure and to concentrate instead on business functionality.

Workflow application logic is one of the largest portions of code that gets rewritten with every application. By using WF, we can chip away the need to rewrite workflows within your applications. Office SharePoint Server Application Services provides a great hosting environment for workflows, so that business users can augment and enhance workflows on their own.

You have seen how we build an enterprise lending solution that can be used as the backbone for all of your lending operations. BizTalk Server allows you to scale and interoperate with many different back ends and external systems, while WF enables your human workflow. This is all tied together using open XML standards from both industry and technology bodies.
More Information

- MSDN Architecture Center (http://msdn2.microsoft.com/en-us/architecture/default.aspx)
- Web Services and Other Distributed Technologies (http://msdn2.microsoft.com/en-us/webservices/default.aspx)

About the Author

Mike J. Walker

Architecture Strategist, Financial Services, Architecture Strategy Team

Microsoft Corp.

Mike Walker is the managing architecture strategist for the Financial Services vertical at Microsoft. Walker is responsible for driving and evangelizing Microsoft’s worldwide industry and vertical strategy in the banking, insurance and securities segments. Specifically, Walker ensures that financial services institutions around the world realize the full extent of Microsoft’s vision and value proposition, by overseeing areas such as industry strategy, marketing, solution development, partner development, thought leadership, and executive relations.

Walker joined Microsoft in early 2006. His background is as a financial services strategist, specializing in business transformation around technology, but he combines this experience with a strong focus on strategic execution. Prior to joining Microsoft, Walker’s worked as an enterprise architect in banking, where he developed business strategies, conducted strategic infrastructure planning, and implemented technology projects. His experience includes bespoke and package implementation, enterprise process design and management, applications management and systems selection.