

B2B Web Services Application

Group Members:

Patricia Zamorski

Julian Tsin

Muhammad Nadeem

Larry Perrone

Project Scope

Project Definition

Building a B2B application for electronic business transactions between an Online Books Seller and a Wholesale Book Supplier with the help of web services invocation.

The Project Scenario

The scenario for this project consists of two fictitious companies:

Tom's Online Books, and

John's Wholesale Book Supplier.

John's company is merely a wholesale supplier of books. He has no storefront and no direct customer sales; he simply provides books at wholesale prices to other book suppliers.

Tom's company runs an online bookstore that allows Internet customers to purchase books through his shopping cart.

Tom carries no inventory; instead, he accepts customer orders and passes them to John, who supplies the books. Because John has no storefront, his only means of generating revenue is to have other bookstores place orders with him. To accommodate this, John has decided to build an ASP.NET application with a Web services layer to allow real-time sales with his vendors, and Tom has decided to increase his productivity and create his own online shopping cart system with ASP.NET and a Web services tie-in to John's application. By making this business collaboration agreement, both hope to increase revenue and productivity.

We'll act as the system designer for both applications. Because the only link is through the Web services layer, providing a generic design allows both applications to be flexible enough for future expansion, and prevents Web service API conflicts between the two development parties.

Some Web Services Scenarios

In this project, no doubt the Web service is the backbone of both applications and is the quintessential components to the B2B collaboration aspect of the products. By implementing asynchronous web service invocations, we can create a fast responding, multithreaded B2B information exchange between two business partners. Without web services, the collaborative requirement of both customers could not have been met.

We will have following three web services in this B2B applications:

- Web service for searching books in John's application

- Web service for creating order in John's application
- Web service for confirming order in Tom's application

Application Architecture Outline:

Use case Modeling

Domain Model

Design Model

Implementation Model

- Database Design
- Database Tier
- Business Tier
- Exceptions and Logging
- Web Services

Data Model

Software Specifications:

ASP.Net

VB.Net

SQL Server 2000

Microsoft Visio

Domain Model

A domain model is illustrated with a set of classes diagrams in which no operations are defined.

It may show

- domain objects or conceptual classes
- association between conceptual classes
- attributes of conceptual classes

Use Cases Model

- Use cases
- Use cases Diagrams

Use Cases

John's Wholesale Store

Information gathered for John's Wholesale Store:

- Must be able to accept orders from other vendors, specifically Tom
- Needs a way to manage his inventory of books
- Needs to be able to view and process orders
- Wants his inventory to be secured from unauthorized vendors
- Wants to provide access to his book inventory to select vendors

By using extrapolation techniques we can identify the following use cases for John's application:

- View books
- Edit book
- View/search orders, with filters for client and order status
- Confirm order (flag an order as completed)

Web service use cases

- Create orders
- Get orders
- Search books (filtered by ID, author name, book name, or availability)

Tom's Online Book Store

Tom has slightly more involved requirements because of the customer interaction. Information gathered for Tom's Online Book Store:

- Requires a public area for regular surfers and a secured area for authorized users
- Needs to be able to track customer accounts
- Must be able to search John's database for books
- Needs a customer login screen
- Needs a shopping cart mechanism to let customers add/remove books from their cart
- Needs an area to let customers review their cart and place an order
- Needs a way to place orders in John's application
- Wants to let customers view the orders they've placed
- Needs to have a way to let John's application notify his system automatically on order completion
- Wants to notify customers by e-mail when their order is completed

- Wants a user friendly error page
- Wants to display to customers the number of items in their cart in an area that's always viewable to the customer

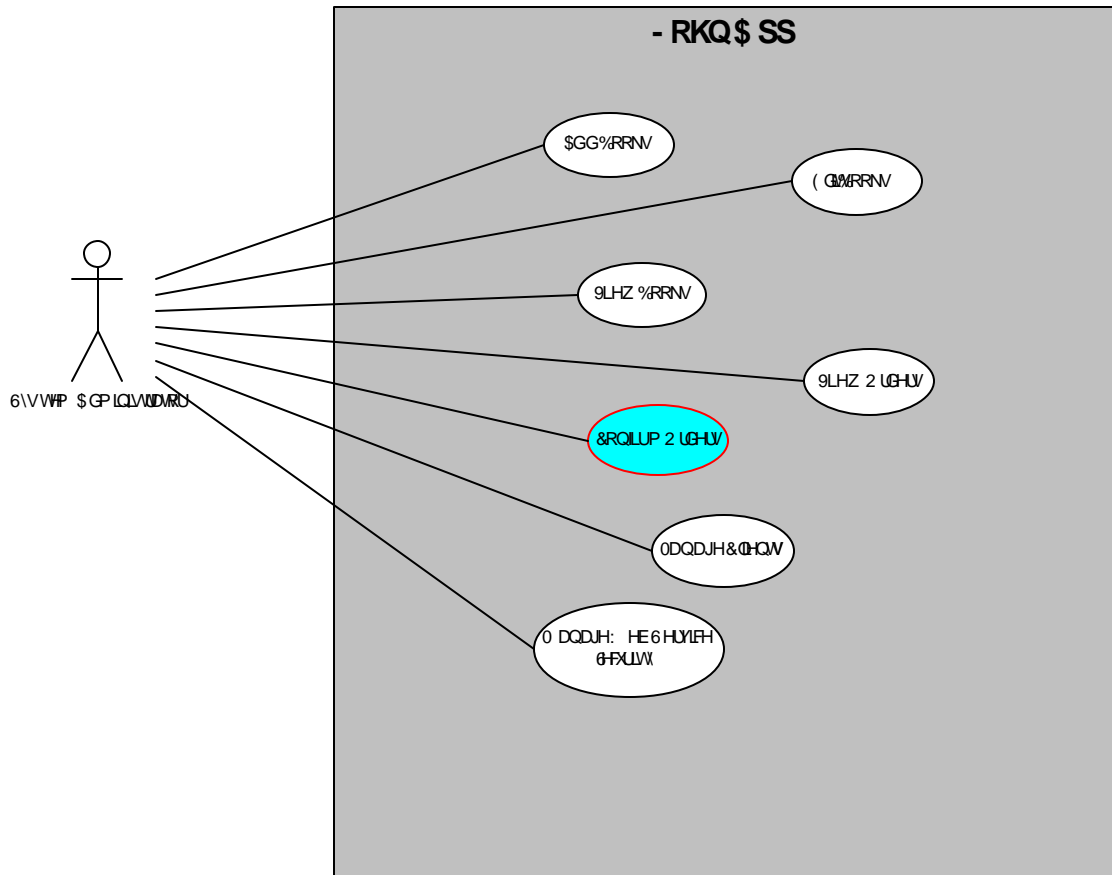
Again, by using our extrapolation techniques we can decipher the following use cases for Tom's application:

- Search books (filter by book name, author, or availability)
- Add book to cart
- Remove book from cart
- View shopping cart (Secured)
- Create order (Secured)
- View customer orders (Secured)
- Create customer
- Log in customer

Web service use cases

- Confirm order

Use Cases Diagrams

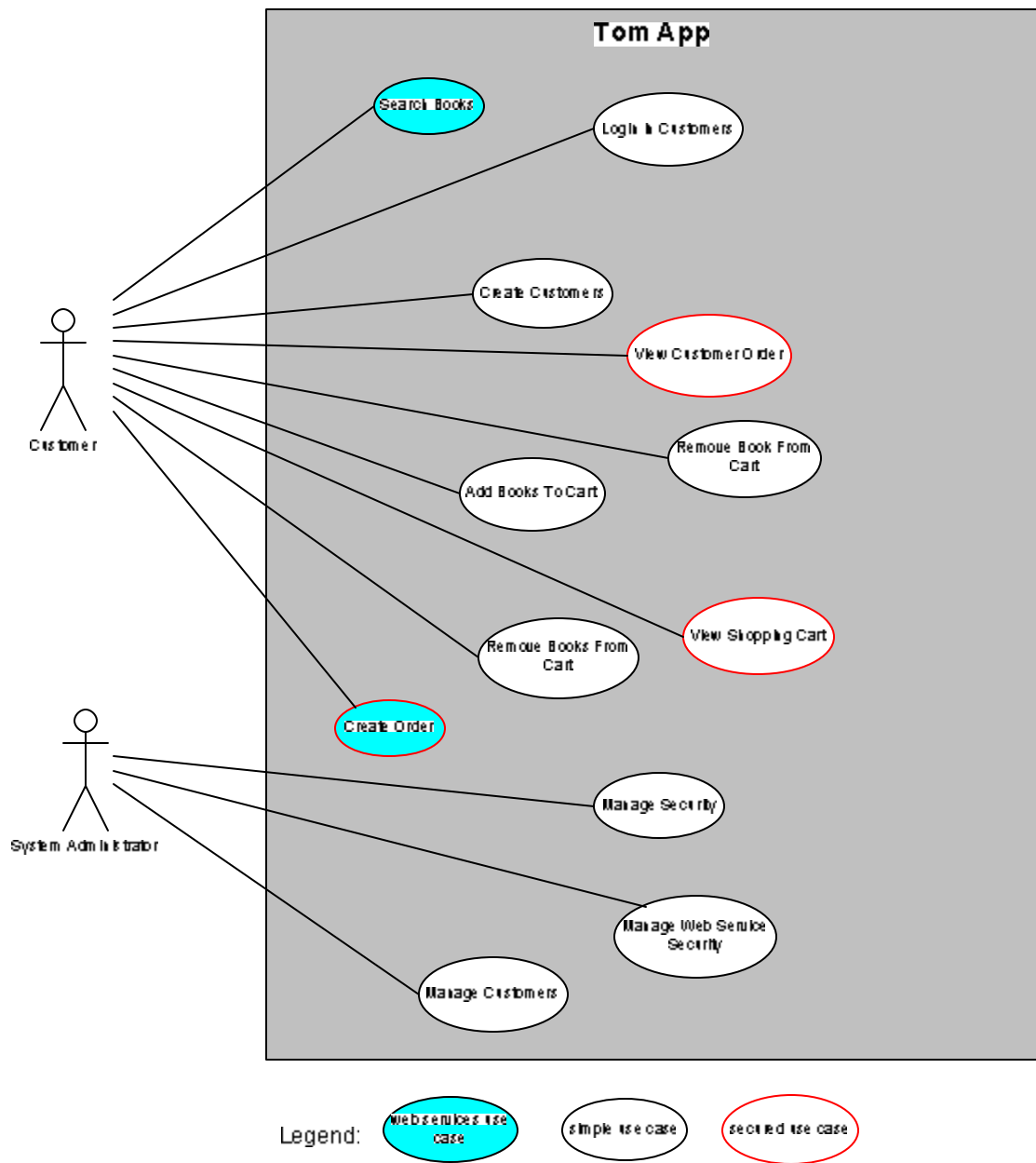


/ HJHQG

ZHEVHVMFHVMH
FDM

VPSO\KVMFDMH

- RKQ\$ SS 8 VH&DMH&RQM VDUJDP



Tom App Use Case Context Diagram

Design Model

Use Case – Search Books

- A customer enters Book Name, Author name, and select AvailabilityID [In Stock, Back Order].
- Listing of books with Author Name and Price is shown for the customer.

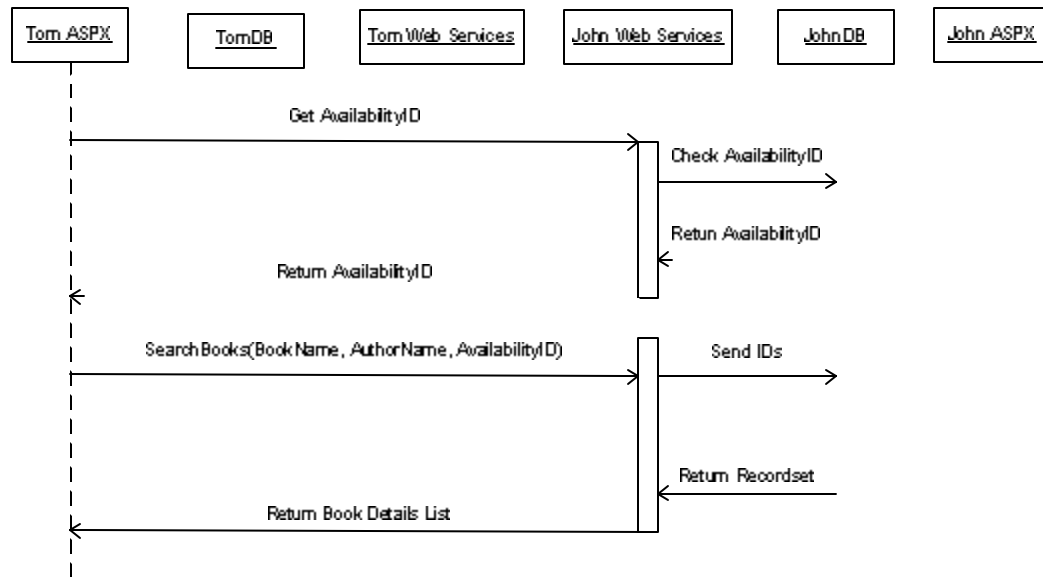
Use Case – Create Order

- **Pre Condition:**
A customer has successfully login.
The customer has added items in shopping cart.
- Customer clicks Make Order button. The system generates OrderID.
- It sends asynchronous call to JohnWS.
- JohnWS creates order data in John Database.
- The Customer receives notification about order creation.

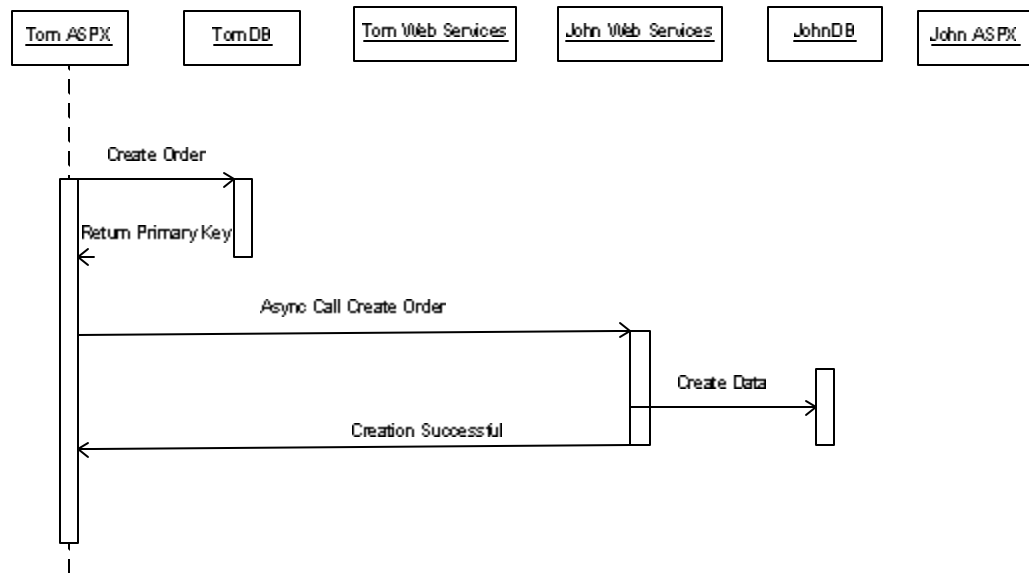
Use Case – Confirm Order

- John App administrator search orders by Clients Name and clicks Confirm Order button.
- System calls TomWS that updates order status in Tom Database.
- TomWS sends notification about order confirmation to JohnApp.

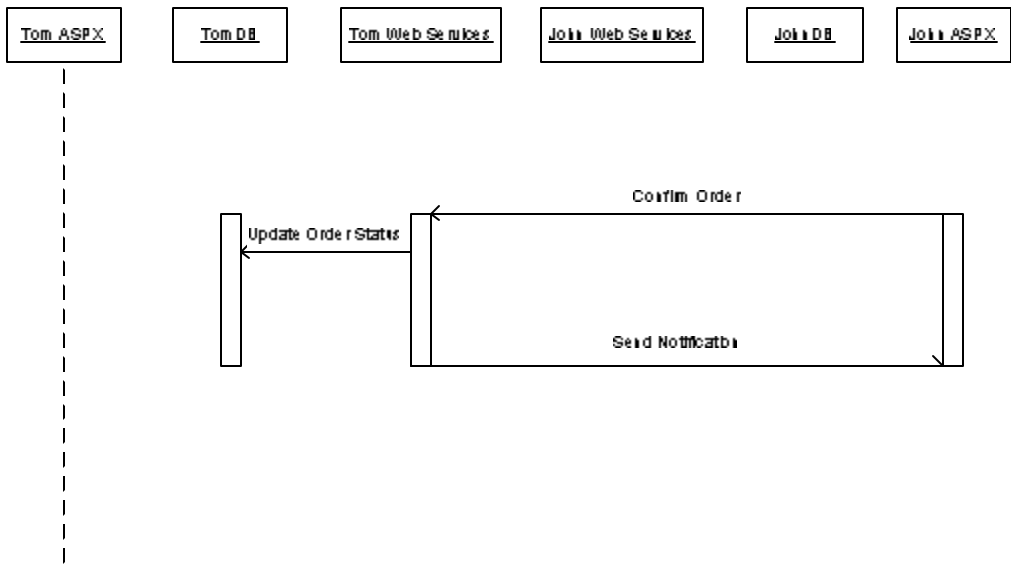
Web Services Sequence Diagram - Search Books [Consumed in Tom App]



Web Services Sequence Diagram - Create Order [Consumed in Tom App]



Web Services Sequence Diagram - Confirm Order [Consumed in John App]



Implementation Model

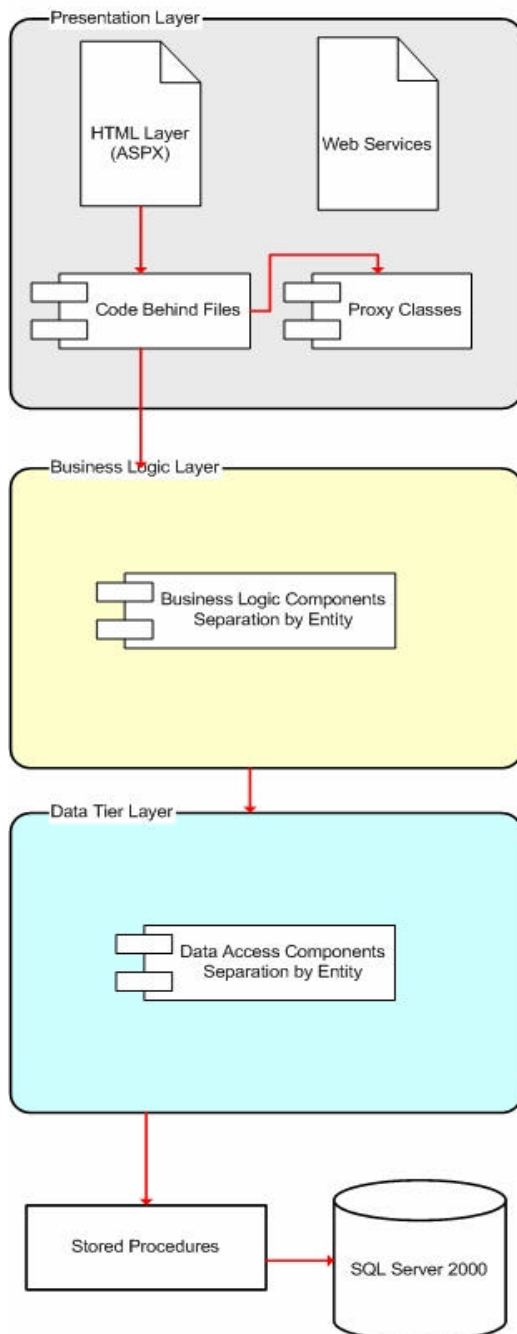
The n-tier architecture model

We used the following product technologies to build our final application:

1. Visual Studio .NET 2002
2. SQL Server 2000.

The application framework employed an n-tier architecture model, a commonly used model in more intricate enterprise systems. A simple diagram of this model can found in figure.

ASP.NET N-Tier Architecture



Exception Handling

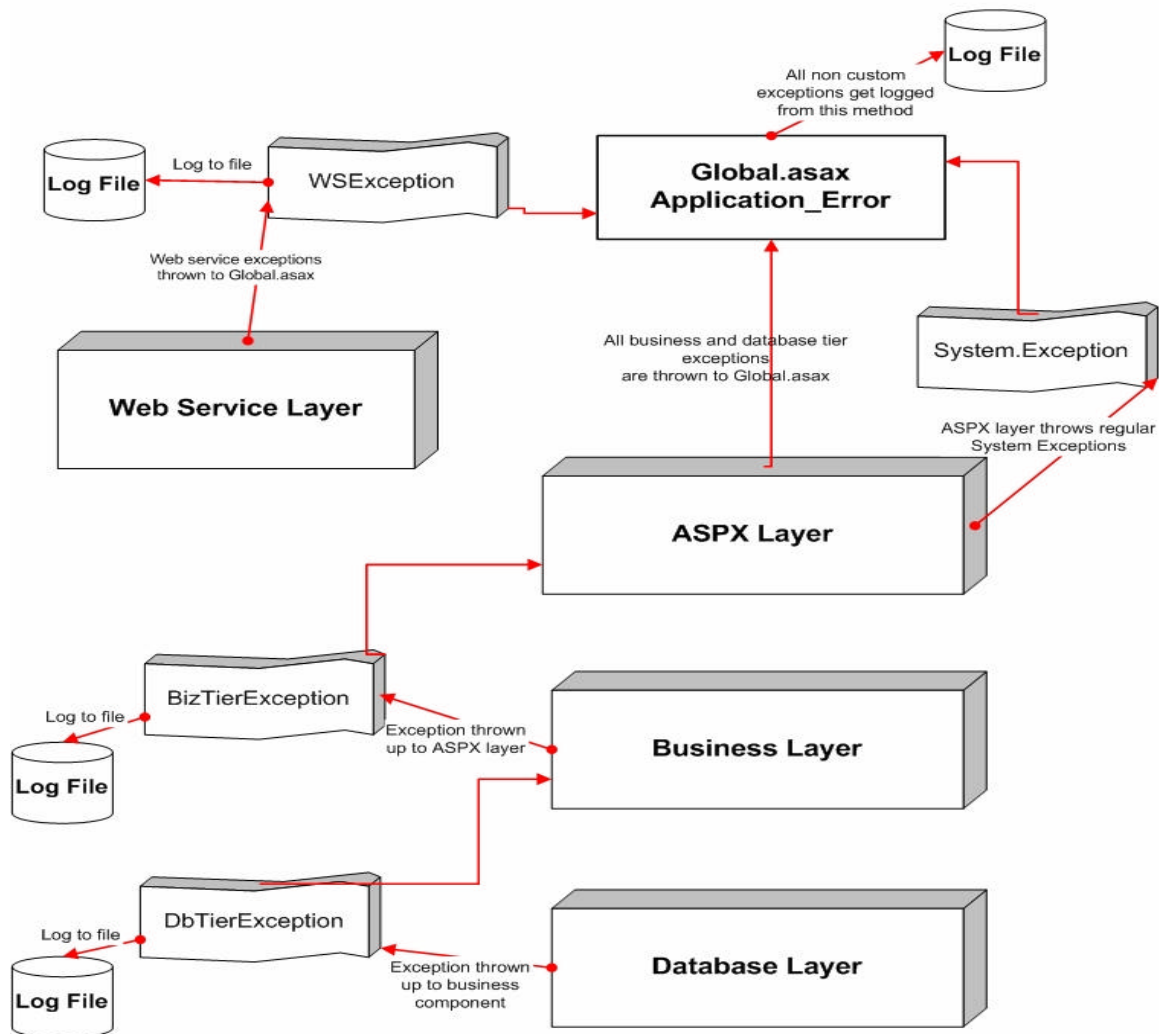
One of the most important aspects of proper system design is exception handling. Each layer is represented by a tier in our framework.

We have a database layer, a business layer, an ASPX layer, and a Web service layer.

By implementing a custom exception class for each layer, we can easily identify and trace our exceptions at any point in our application.

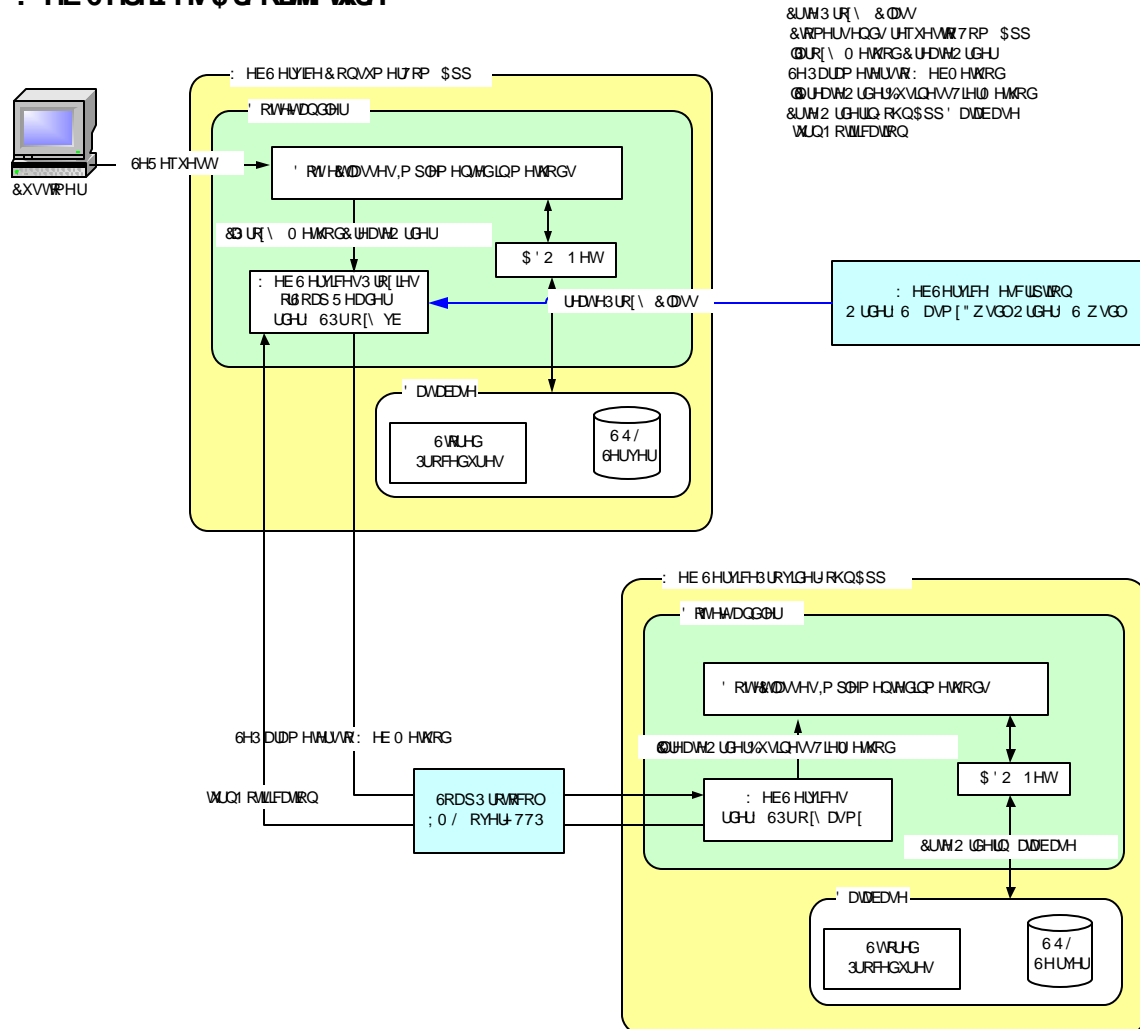
The figure diagrams the exception flow in both of our applications

Application Exception Flow in n-Tier Design



A diagram shows the web service mechanism for creating an order.

: HE 6 HUYLFHV\$ UFKLMFVXUH

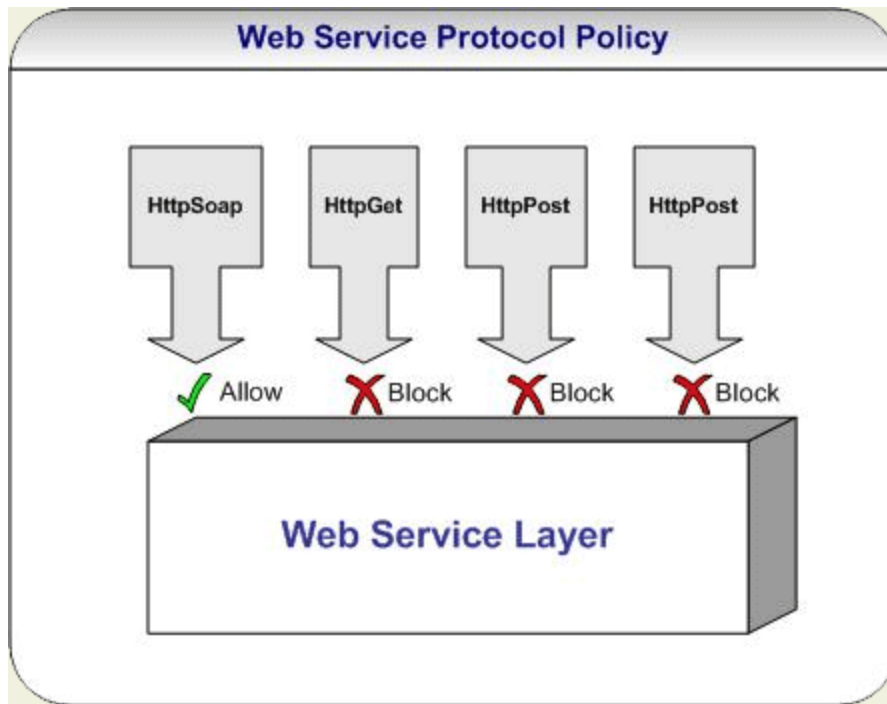


Web Service Security

Security is probably the biggest technology-related buzzword. There are two main aspects of application security: authorization/authentication and data transmission. You basically want to keep out the people who shouldn't be accessing your application and also prevent data from being captured in transit.

To secure John's Web services, we have implemented a custom SOAP header, which is essentially an inner class definition with some public properties that get wrapped into the SOAP envelope during a Web method call. By defining properties of this SOAP header class, we have set properties that allowed the Web service to determine if the consumer is in fact authorized to access the application.

The second security measure we have implemented is the lock down of specific protocols, thus preventing the Web service from being accessed in certain ways. Following figure depicts the protocol policies for John's application.



Web Service Protocol Policy Diagram

By removing the HttpPost, HttpGet, and Documentation protocols, we have prevented the invocation of our Web services via Post and Get, and also remove the ability to generate WSDL documents, which could allow unauthorized users to generate Web references and proxy classes for our Web services.

By using both security measures in conjunction, we accomplish the following:

- We prevent the discovery and the ability to invoke our Web service through any means other than HttpSoap, which essentially means the consumers must have a valid proxy class prior to the Web service lock down. This allows the developer to be the only distributor of Web service access.
- If, somehow, a user managed to discover the Web service, he or she would not be able to generate the WSDL file for the service because the Documentation protocol is removed.
- If, somehow, an unauthorized user managed to acquire a proxy class or WSDL file by some extraneous means, he or she would not pass the authentication checks because the user wouldn't have valid ClientIds or WSTokens, which must be present in our custom SOAP header.

We have a nearly impenetrable security mechanism in place. An unauthorized user must break through three strict tiers of security before he or she would be able to compromise our application. Adding SSL to the equation only further increases the security level.

Web service for searching books in John's application

This web service searches books in John's application and returns list of books to Tom's application.

Code implementation of searching books

```
#####
' When the search button is clicked we invoke John's web service and search his books. The result is bound to our datagrid
##### Private Sub
btnSearch_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnSearch.Click
    Dim ProxyObj As New BookWSProxy.BookWS()
    ProxyObj.BookSecurityContextValue = WSUtil.GetBookSecurityContext

    Dim AvailabilityId As Int32
    If Not WSB2BUtil.Utils.IsNull(Me.cboAvailability.SelectedItem.Value) Then
        AvailabilityId = Convert.ToInt32(Me.cboAvailability.SelectedItem.Value)
    End If

    Dim BookDs As DataSet = ProxyObj.SearchBooks(Nothing, Me.txtAuthor.Text, Me.txtBookName.Text, AvailabilityId)
    If Not BookDs Is Nothing Then
        Me.dgBooks.DataSource = BookDs.Tables(0)
        Me.dgBooks.Visible = True
        Me.DataBind()
    End If
End Sub
```

Web service for creating order in John's application

This web service creates order for the books in John's application.

Code implementation of creating order

```
#####
' The customer clicks this button to create their order. We create the order in this system first, generating the Pk
' for the order, and then invoke John's WS to create the order in his system. If the webservice fails, we delete
' the order from our system. John's web service is invoked asynchronously and the user is notified via email regarding
' the status of the order
#####
Private Sub btnCreateOrder_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles btnCreateOrder.Click
    Dim OrderObj As New BizTier.OrderServices()
    Dim SessionObj As ClientSession = Session.Item(ClientSession.CLIENT_SESSION_REF)

    Dim OrderId As Int32 = OrderObj.CreateOrder(SessionObj.CustomerId)

    Try
        Dim ProxyObj As New OrderWSProxy.OrderWS()
        ProxyObj.OrderSecurityContextValue = WSUtil.GetOrderSecurityContext
        ProxyObj.BeginCreateOrder(OrderId, SessionObj.BookIds.ToArray(), New AsyncCallback(AddressOf ReceiveResponse),
ProxyObj)
        Me.lblMsg.Text = "Thank you for your order"

    Catch ex As System.Web.Services.Protocols.SoapException
    Catch ex As Exception
        ' If the web service fails for any reason, we must remove the order from the Db
    End Try
```



```

        OrderObj.DeleteOrder(OrderId)
        Me.lblMsg.Text = "There was an error processing your order"
    Finally
        'Blank the datagrid of the order that was placed
        Me.dgCart.DataSource = Nothing

        ' Clean up the order information contained in the session
        SessionObj.BookIds = New ArrayList()
        SessionObj.NumItems = 0

        Me.DataBind()
    End Try

End Sub

```

Web service for confirming order in Tom's application

For Tom, only one Web service is required. Its purpose is to provide a way for John's application to notify Tom's application of completed orders because he needs only to expose a mechanism for John's application to confirm orders. Access to Tom's Web service will be provided by distributing a proxy class to John's application.

Code implementation of confirming order

```

#####
' Confirms a pending order by invoking Tom's web service and updating the order in his application first.
' The call is made asynchronously to allow for fast user response
#####
Public Sub Confirm_Order(ByVal sender As Object, ByVal e As DataGridCommandEventArgs)
    ' Grab the Id of the order being confirmed
    Dim OrderId As Int32 = Convert.ToInt32(e.Item.Cells(0).Text)
    Dim OrderObj As New OrderServices()

    'Get the information regarding the order
    Dim OrderDs As DataSet = OrderObj.GetOrders(OrderId, Nothing, Nothing)

    '## NOTE ##
    ' If we add additional clients, we need to check the client Id and invoke
    ' the correct web service method for that particular client. Since
    ' Tom is the only client in our example, we invoke his web service directly

    Dim OrderProxy As New OrderWSProxy.OrderWS()
    OrderProxy.OrderSecurityContextValue = WSUtil.GetOrderSecurityContext()
    OrderProxy.BeginConfirmOrder(OrderId, OrderDs, New AsyncCallback(AddressOf CompleteOrder), OrderProxy)

    Me.lblMsg.Text = "The order has been sent for completion processing"
    Me.lblMsg.Visible = True
    Me.dgResults.Visible = False
End Sub

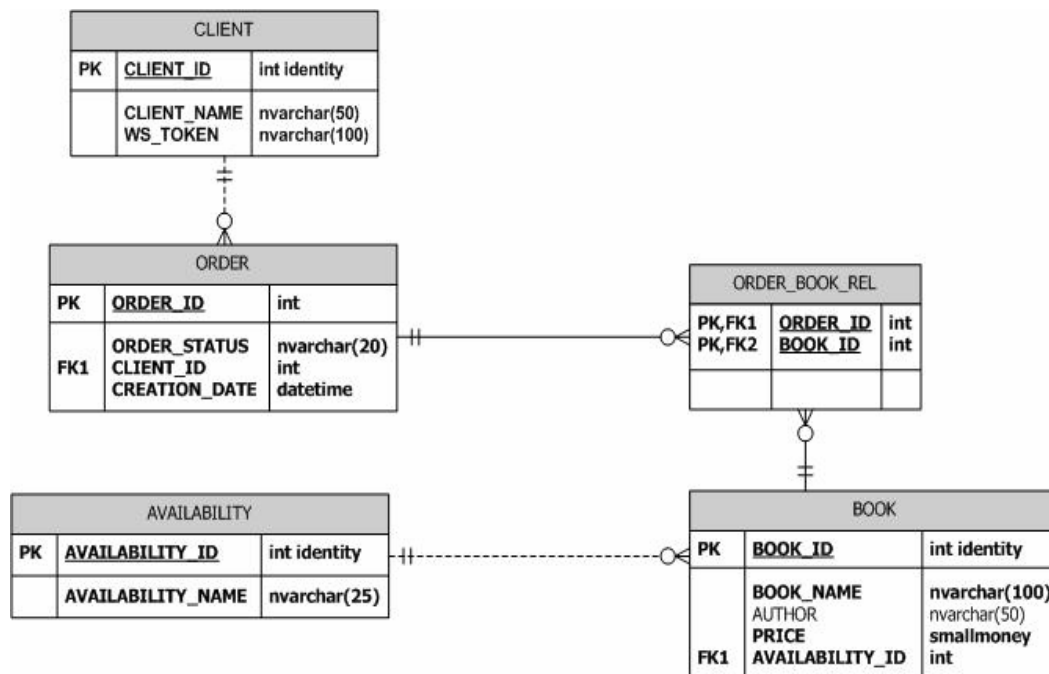
```

Asynchronous Behaviour of Web Services

Data Model

John's App:

- **Client:** Table to store clients that are capable of placing orders with John.
- **Order:** Stores the details about placed orders.
- **Book:** Stores John's inventory of books.
- **Availability:** Support table storing a list of availabilities. This table was added by normalizing the Book table and removing the redundant availability text field and replacing it with an optimized foreign key integer field to another table.
- **Order_Book_Rel:** Many-to-many relationship table between books and orders. This is required because many books can be tied to many orders. By simply adding book IDs and order IDs to this table we remove all redundancy required to tie orders to books.



Tom's App:

- **Order.** Used as a singular entity to map an entire order to a customer. The details of the order are stored in John's application.
 - **Customer.** Stores a list of customers who can log into Tom's application and place orders.
- By adding some basic data fields to each of the tables above we have our completed database structure

