

Brücken bauen mit Metadaten, Teil 2

Das Information Bridge Framework verbindet Web Services und Office 2003

von Marcel Gnoth

Mit dem Information Bridge Framework, kurz IBF, erhalten .NET-Entwickler die Gelegenheit, Unternehmensdaten „intelligent“ in eine Office-2003-Anwendung zu übernehmen. Nachdem in Ausgabe 10.2005 des *dot.net magazin* das Information Bridge Framework eingeführt wurde, soll in diesem Teil die Entwicklung einer IBF-Lösung beschrieben werden.

IBF bietet eine standardisierte Schnittstelle, mit der ohne „große Programmierung“ Daten von Web Services oder auch beliebigen Assemblies abgerufen werden können. Diese Daten werden dem Office-Anwender im Aufgabenbereich (auch *TaskPane* genannt) präsentiert. Die Web Services können direkt von einer LOB-Applikation zur Verfügung gestellt oder aber mit Visual Studio entwickelt werden. Im folgenden Szenario wird der Web Service mit Visual Studio entwickelt. So wird deutlich, welche Bedingungen ein Web Service mitbringen sollte, damit die IBF-Entwicklung möglichst reibungslos verläuft. Falls die LOB-Applikation ein anderes API als Web Services anbietet, müssen die Web Services dann vom IBF-Entwickler implementiert werden.

Das Szenario

Als Erstes muss definiert werden, welche Aufgaben die IBF-Lösung übernehmen soll. Für dieses Beispielszenario soll ein

kurz & bündig

Inhalt

Im zweiten Teil des Überblicks geht es um die konkrete Entwicklung von Projekten mit dem IBF und Visual Studio

Zusammenfassung

Visual Studio unterstützt die Entwicklung von IBF-Anwendungen durch eine Projektvorlage und einen Metadaten-Explorer

Smart Tag eine KundenID, wie *ALFKI*, aus der Northwind-Datenbank erkennen und als IBF-Smart-Tag kennzeichnen. Der Anwender kann dann über das Smart Tag Informationen zu diesem Kunden in der *TaskPane* einsehen. Am Anfang steht die Definition der Artefakte, das sind die Bestandteile der IBF-Lösung.

Entity und Views

Entitäten sind Geschäftsobjekte, wie zum Beispiel ein Kunde, eine Bestellung oder ein Ansprechpartner. Für die Anzeige einer Entität muss mindestens eine *Default-View* definiert werden. Eine *View* kann Adresse und Ansprechpartner des Kunden anzeigen, eine andere *View* kann Zahlungsmodalitäten des Kunden darstellen. Diese *Views* werden in XML definiert (Listing 1), so wie die Daten nachher auch von Web Services übertragen werden.

Reference und Relationship

Eine Referenz ist eine Art Primärschlüssel für den Zugriff auf eine Entität. Über eine Referenz können die Daten für eine *View* abgerufen werden. In den IBF-Metadaten werden Referenzen von *ViewLocator*-Elementen verwendet. Für eine Entität können mehrere Referenzen definiert werden. Ein Kunde kann über seine ID, aber auch über seine E-Mail-Adresse gefunden werden (Listing 2).

Über Relationships lassen sich die *Views* verschiedener Entitäten miteinander ver-

knüpfen. Der Anwender kann dann in der *TaskPane* zu den Büchern eines Autors oder dem Autor eines Buches navigieren, wenn die *AuthorsView* und die *BooksView* über eine *AuthorID*-Referenz miteinander verbunden sind (Abbildung 1). Dabei können die Daten für die *Views* über unterschiedliche Web Services abgerufen werden.

Listing 1

Eine einfache View mit XML-Format definiert ein Kunden- und ein Buchobjekt

```
<Customer xmlns="NTeam-Data">
  <CustomerID></CustomerID>
  <CompanyName></CompanyName>
  <ContactName></ContactName>
  <ContactTitle></ContactTitle>
  <City></City>
  <Region></Region>
  <Phone></Phone>
</Customer>
<Book xmlns="NTeam-Data">
  <ISBN></ISBN>
  <Title></Title>
  <SuggestedPrice></SuggestedPrice>
</BookView>
```

Listing 2

Eine Referenz auf eine Identität

```
<CustomerID ID="ALFKI" xmlns="NTeam-Data" />
<CustomerEmailAddress="Chef@Alfki.de" xmlns="
  NTeam-Data" />
<BookIDReference ISBN="0972317937" xmlns="
  NTeam-Data" />
```

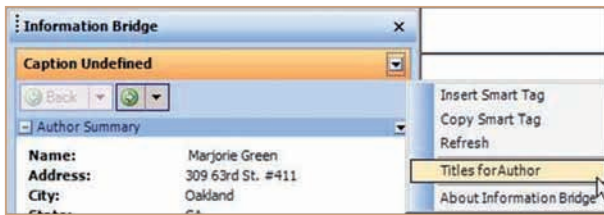


Abb. 1: Über eine Relationship können die Bücher eines Autors abgerufen werden

Listing 3

Die Serialisierung der Klasse wird über Attribute gesteuert

```
[XmlRoot("Customer", Namespace="NTeam-Data",
        IsNullable=false)]
[XmlType("Customer", Namespace="NTeam-Data")]
public class Customer {
    // Customer ID
    [XmlElement]
    public string CustomerID {
        get{return this._CustomerID;}
        set{this._CustomerID = value;}
    }
    // Company Name
    [XmlElement]
    public string CompanyName {
        get{return this._CompanyName;}
        set{this._CompanyName = value;}
    }
}

[XmlRoot("CustomerID", Namespace="NTeam-Data",
        IsNullable=false)]
[XmlType("CustomerID", Namespace="NTeam-Data")]
public class Customer_ID {
    // Fields
    private string _CustomerID;
    // Customer ID Attribute
    [XmlAttribute]
    public string CustomerID {
        get{return this._CustomerID;}
        set{this._CustomerID = value;}
    }
}
```

Listing 4

```
[WebMethod (Description="Get-Method for
                Customer-View")]
public Customer GetCustomer(Customer_ID
                objCustomerID)
{ ... }
[WebMethod (Description="Put-Method for
                Customer-View")]
public void PutCustomer(Customer objCustomer)
{ ... }
[WebMethod (Description="Act-Method for Changing
                Order-State")]
public void ChangeOrderStatus(OrderID orderID,
                OrderStatus status)
{ ... }
```

Erstellen der Klassen für Web-Service-Parameter

Die Web Services, die von IBF aufgerufen werden, sollen die zuvor definierten XML-Daten verarbeiten können. Dafür bietet es sich an, eigene Geschäftsklassen zu implementieren, die serialisiert, genau solche XML-Strukturen erzeugen. Diese Klassen können auch Zugriffslogik auf die Datenbank oder das LOB-System enthalten. Objekte dieser Klassen werden dann als Parameter für die Webmethoden verwendet. Über Attribute wird die Serialisierung der Klassen durch den SOAP-Formatter gesteuert (Listing 3).

Web Services

IBF kann mit beliebigen Web Services zusammenarbeiten. Allerdings erwartet ein generisches Framework wie IBF gewisse Standardwege für den Zugriff. Bieten Web Services eines LOB-Systems keine IBF-konformen Webmethoden, so müssen die XML-Daten über Transformationen angepasst werden. Es werden drei Arten von Webmethoden unterschieden. *Get*-Methoden bekommen eine Referenz als Parameter übergeben und liefern Daten für eine *View* zurück. *Put*-Methoden aktualisieren eine Entität im LOB-System mit geänderten *View*-Daten. *Act*-Methoden können Aktionen im System auslösen und gehen über das Lesen und Schreiben eines Objektes hinaus. So kann zum Beispiel der Bestellstatus geändert werden, was im LOB-System vielleicht mehrere Prozesse auslöst. Zu guter Letzt lohnt es sich, Testmethoden zu implementieren. Da die Web Services komplexe Datentypen als Parameter übergeben bekommen, können sie nicht im Internet Explorer aufgerufen werden. Deshalb bieten sich Testmethoden an, die einfache Parameter vom Typ *Numeric* oder *String* haben und die dann ihrerseits die echten Webmethoden aufrufen und deren Ergebnis zurückliefern (Listing 4).

Ran an die Metadaten mit Visual Studio

Nachdem der Web Service steht, geht es nun an die IBF-Entwicklung. In Visual Studio .NET 2003 finden Sie nach der Installation des IBF SDK einen neuen IBF-Projekttyp. Fügen Sie zu Ihrer Web-Service-Projektmappe ein IBF-Projekt hinzu. Das Projekt besteht nur aus einer einzigen XML-Datei namens *MSIBF-Metadata.xml*, die im Projekt-Explorer zu finden ist und die Sie auch direkt bearbeiten können. Da das aber sehr umständlich ist, gibt es den *Metadata Explorer* als neues Fenster in Visual Studio – über ihn können Sie Artefakte anlegen und bearbeiten. Das ist aber immer noch aufwendig und daher gibt es seit der Version 1.5 auch einige Wizards, die bei Routineaufgaben helfen. Diese Wizards können über das *Metadata-Guidance*-Fenster gestartet werden (Abbildung 2). Es gibt im Projektverzeichnis zwei Unterverzeichnisse, die je eine XML-Datei enthalten. Unter *Current Metadata* sind die gerade am Client bearbeiteten Metadaten zu finden – *Original Metadata* enthält die vom Server gelieferten. Im *Metadata Explorer* ist eine Schaltfläche zum Publizieren der Metadaten zum Server zu finden – danach sind beide XML-Dateien synchronisiert.

Import der Metadaten

Die XML-Datei des Projektes enthält am Anfang nur ein paar Header. Als Erstes

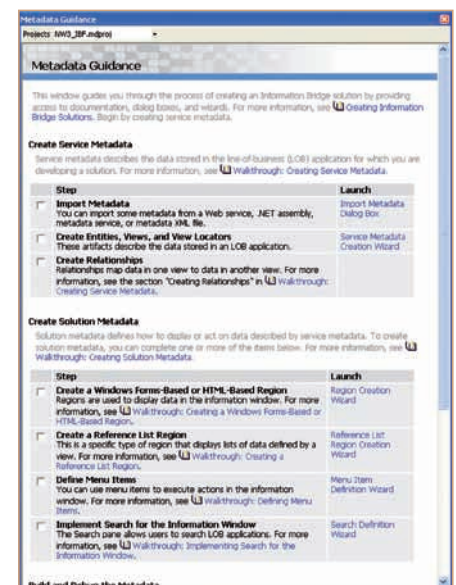


Abb. 2: Das Metadata-Guidance-Fenster bietet dem Entwickler Unterstützung

werden nun zwei Mengen von Metadaten über das *Metadata-Guidance*-Fenster importiert. Über die *Import Metadata Toolbox* können XML-Metadaten aus unterschiedlichen Quellen zu den bestehenden Metadaten hinzugefügt werden. Im ersten Schritt wird das WSDL des Web Service importiert. Beim Import wird gefragt, ob die Daten mit den vorhandenen zusammengeführt oder die vorhandenen ersetzt werden sollen. Im Regelfall werden die Daten gemischt. Anschließend werden weitere Metadaten aus einer Datei namens *Microsoft.InformationBridge.Framework.UI.InformationBridgeScope.Metadata.xml* des *IBF Ressource Kit* importiert. Diese Daten enthalten Rumpfe für die wichtigsten Artefakte. Nach dem Import der Daten sind im *Metadata Explorer* eine ganze Reihe von neuen Einträgen zu finden. So existieren zum Beispiel jetzt Ports für den Zugriff auf die Web Services und Schemata, die dem XML der Parameter der Webmethoden entsprechen – jetzt kann die eigentliche Entwicklung beginnen.

Erstellen einer Entity

Zur Anzeige eines Kunden müssen einige weitere Artefakte definiert werden. Das sind *Entity*, *View* und *ViewLocator*. Zum Glück gibt es dafür jetzt den *Service Metadata Creation Wizard* (Abbildung 3). Hier werden in wenigen Schritten neue Artefakte erstellt und mit den dazugehörigen Schemata verknüpft. Der *ViewLocator* wird mit einer Operation verknüpft, die den Web-Service-Aufruf durchführt. Diese Operation wurde durch den Import der Web-Service-Daten erstellt.

Region Creation Wizard

Nachdem die Daten definiert wurden, geht es nun um die Darstellung in einer Region. Der Wizard (Abbildung 4) fragt, welche *View* dargestellt werden soll. Über ihn kann auch eine *Action* zur Anzeige der *View* in der Region erstellt werden, die aus mehreren Operationen besteht. Die Daten können als HTML oder aber als *WinForm UserControl* angezeigt werden. Der Wizard erstellt dann ein neues Projekt in der Solution mit einem *UserControl*.

Dabei taucht bei einem nicht englischen Framework ein kleines Problem auf. Der Wizard verwendet das Tool *Xsd.exe* und wartet auf die englische Textausgaben des Tools. Tja, und bei einer lokalisierten deutschen Version hat der Wizard Pech, da nur deutsche Texte zurückgegeben werden. Als schnelle Lösung kann die Ressourcendatei mit den deutschen Texten umbenannt werden, dann findet *Xsd.exe* sie nicht mehr und verwendet die inkompilierten englischen Texte und alles läuft prima. Die Datei ist unter *...\Programme\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin\DE\xsd.resources.dll* zu finden. Abschließend erhält das Projekt noch einen starken Namen und muss kompiliert werden. Damit die DLL innerhalb von Office laufen darf, muss die *Code Access Security* für diese Assembly auf *Full Trust* gesetzt werden.

Die Ernte der Früchte

Ganz fertig ist die Lösung noch nicht, aber Sie können jetzt eine erste Kostprobe Ihrer IBF-Lösung genießen. Der *Region*

Anzeige



Abb. 3: Der Service Metadata Creation Wizard

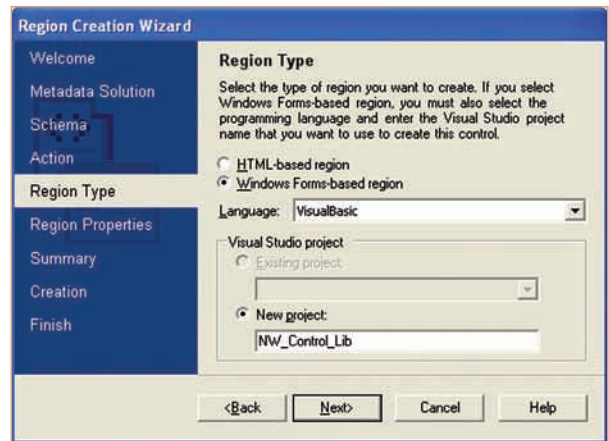


Abb. 4: Der Region Creation Wizard erstellt neue UserControl's zur Datenanzeige

Creation Wizard hat eine *Action* erstellt, die Customer-Daten anzeigen kann. Zu finden ist die *Action* im *Metadata Explorer* unterhalb von ENTITY|VIEW|ACTION. Über das Kontextmenü kann die *Action* getestet werden und im Visual Studio erscheint eine frei schwebende *TaskPane*, die Daten des Customer anzeigt. Als Parameter muss der entsprechende *ViewLocator* und eine Referenz in XML angegeben werden, genauso wie zuvor die Referenz definiert wurde.

Smart Tags

Abgesehen vom Web Service, der aber nicht zum IBF-Projekt gehört und in der Praxis vom LOB-System zur Verfügung

Dem IBF auf den Zahn geföhlt

Das IBF ist als Entwicklerangebot von Microsoft schwer einzuschätzen. Offiziell steht es unter msdn.microsoft.com/office/understanding/ibframework/downloads/kostenloszumDownload zur Verfügung – wie es sich aber mit den Lizenzkosten für den Einsatz in der Praxis verhält, muss vermutlich mit der Microsoft GmbH im Detail geklärt werden. Zu diesem wichtigen Punkt kann der Autor daher leider keine Angaben machen. Nicht weniger einfach ist, zu entscheiden, ob Unternehmensdaten nun mit dem IBF oder mit einer alternativen Technik, wie den Visual Studio Tools for Office realisiert werden. Auch wenn es sich hier um den klassischen Vergleich von Äpfel und Birnen handelt, kommt am Ende das Gleiche dabei heraus. Das IBF kann seine Vorteile ausspielen, wenn Daten tatsächlich über Web Services abgefragt werden und nach dem Prinzip „Put, Get, Act“ arbeiten. Ein Nachteil ist, dass sich Entwickler trotz gewisser Hilfen gut mit XML und vor allem XSLT auskennen müssen. Am Ende wird es eine Frage der Vorlieben und Entwicklerkompetenz sein. In Zukunft werden auch andere Hersteller IBF-Technologien benutzen und eigene Lösungen anbieten.

gestellt wird, mussten Sie keine einzige Zeile Code schreiben – das *UserControl* wurde ja vom Wizard erstellt. Aber nun ist doch ein wenig Programmierarbeit vonnöten. IBF-Aktionen werden meist über ein Smart Tag aufgerufen und dieses müssen Sie selbst programmieren – das ist aber nicht sonderlich schwer. Bei der IBF-Client-Installation wurde ein generisches IBF-Smart-Tag registriert, das die Brücke zwischen Office und Ihren Smart Tags herstellt. Dadurch wird die Entwicklung Ihres IBF-Smart-Tag einfacher und Sie müssen es auch nicht auf dem Client registrieren. Stattdessen wird das Smart Tag in die Metadaten eingetragen. Es sind nur zwei Interfaces zu implementieren. *IRecognizer* prüft Text auf definierte Schlüsselwörter und kann dann das Wort als Smart Tag markieren. *IActionHandler* definiert die Aktionen und Kontextmenüeinträge eines Smart Tag und muss nicht unbedingt implementiert werden. Ohne *IActionHandler* steht nur die Standardaktion *ShowDetails* des Smart Tag zur Verfügung.

Debuggen, Publishing und Deployment

Über die Eigenschaften des IBF-Projektes können Sie festlegen, ob Sie die Metadaten oder Managed Code debuggen möchten – beides zusammen geht leider nicht. Sie können Haltepunkte auf den Operationen im Metadata Explorer oder im Code des *UserControl* setzen. Ist die Lösung fertig, müssen die Metadaten zum Server geschickt werden – von dort können alle Clients sie abrufen. Assemblies, wie das *UserControl* oder das Smart Tag können auf einem Webserver im Intranet abgelegt werden, von wo sie dann bei Bedarf von

den Clients heruntergeladen werden. Auf den Clients muss „nur“ die *Code Access Security* entsprechend konfiguriert sein.

Fazit

Der Artikel hat einen abgerundeten Überblick über die IBF-Entwicklung gegeben, die ohne eine grundlegende Einführung für die meisten interessierten Entwickler nur schwer fassbar sein dürfte. Eine ausführliche Schritt-für-Schritt-Beispielanwendung (allerdings in Englisch) finden Sie unter [1] und [2]. Die Vortragsfolien und Beispiele der IBF-Session des Autors auf der BASTA! 2005 finden Sie unter [3]. Außerdem sprach der Autor im Oktober 2005 in insgesamt zehn deutschen Städten im Rahmen einer TechTalk-Veranstaltung rund um die Office-Programmierung im .NET-Zeitalter [4].

Marcel Gnoth führt, neben seiner Arbeit als Software-Ingenieur, Trainings durch, publiziert in renommierten .NET-Fachmagazinen und tritt, so oft er kann, als Sprecher auf der BASTA! auf. Seine Schwerpunkte liegen bei verteilten Informationssystemen und der Office-Integration. Kontakt: marcel@gnoth.net und www.gnoth.net.

Links & Literatur

- [1] The Code Project – Information Bridge 1.5 Walkthrough – Part 1 – C# Programming: www.codeproject.com/csharp/IBFWalkthroughPart1.asp
- [2] The Code Project – Information Bridge 1.5 Walkthrough – Part 2 – C# Programming: www.codeproject.com/csharp/IBFWalkthroughPart2.asp
- [3] Folien der IBF-Session auf der BASTA! 2005: www.gnoth.net/BASTA/Feb02.htm
- [4] www.techtalk.ms