

The Information Bridge Framework 1.0

Title: The Information Bridge Framework 1.0

Subtitle: The Bridge between MS Office 2003 and Line-Of-Business (LOB) applications.

Authors: Wim Uyttersprot, Patrick Tisseghem, Gert Servranckx, Jan Tielens (U2U)

Traditional Line-Of-Business (LOB) systems that process operational data are completely different from desktop systems used by information workers who are using Microsoft Office. Both environments have a common goal: support business processes, but they have a different line of approach. The Information Bridge Framework (IBF) allows us to bridge the gap between these two worlds. IBF is a new framework, particularly designed for the information worker that uses more than one Line-Of-Business application, but who also wants to work with Microsoft Office 2003 simultaneously. In particular IBF is exceptionally useful in organizations where there is a need to open business applications and processes to information workers, from within Microsoft Office smart clients.

Positioning IBF

How is business data efficiently processed today when for example that data arrives by email? A lot of us will answer: by copying data from the email to the business application. How does operational data get extracted from Excel and Word documents? Unfortunately too many times the answer is: manually.

The Microsoft Information Bridge Framework bridges that gap between Microsoft Office 2003 and the different business applications used by the information worker. Microsoft has designed IBF to be extendable and scalable in an enterprise environment. The framework can be used to develop a new breed of applications that connect LOB-systems with Microsoft Office in a smart way.

IBF allows retrieving, analyzing and managing business data from within Microsoft Office smart clients. In other words by using IBF, business information and corresponding methods stay defined within the enterprise environment and still can be exposed by web services inside the context of Office documents. So information workers can manage business data efficient and centralized following the procedures defined in business applications.

IBF version 1.0 works with Word 2003, Excel 2003 and Outlook 2003. In version 1.1 additional metadata designers will be added, as well as support for InfoPath 2003. In future releases support for Microsoft Windows SharePoint Services is expected, and finally IBF will become an integral part of the new client desktop operating system, code named 'Longhorn'.

IBF offers a standardized approach that's based on creating metadata definitions. Developers of LOB-applications create business objects and services offered by these business objects by using web services. The solution developers define which web services will be consumed by smart clients. They bond the business objects and create associations with the user interface of the Microsoft Office environment. Finally information workers use the business object from within the context of their Office documents that rely on smart tags, XSD enriched documents and a new task pane.

A typical business scenario

The power of IBF is revealed in a typical business scenario: an information worker would like to consult business data of a customer from who the company name occurs in a Word document. In figure 1 the company name "Adventure Works" is recognized by a smart tag. The information worker can use this smart tag to retrieve business data and show it in a task pane.

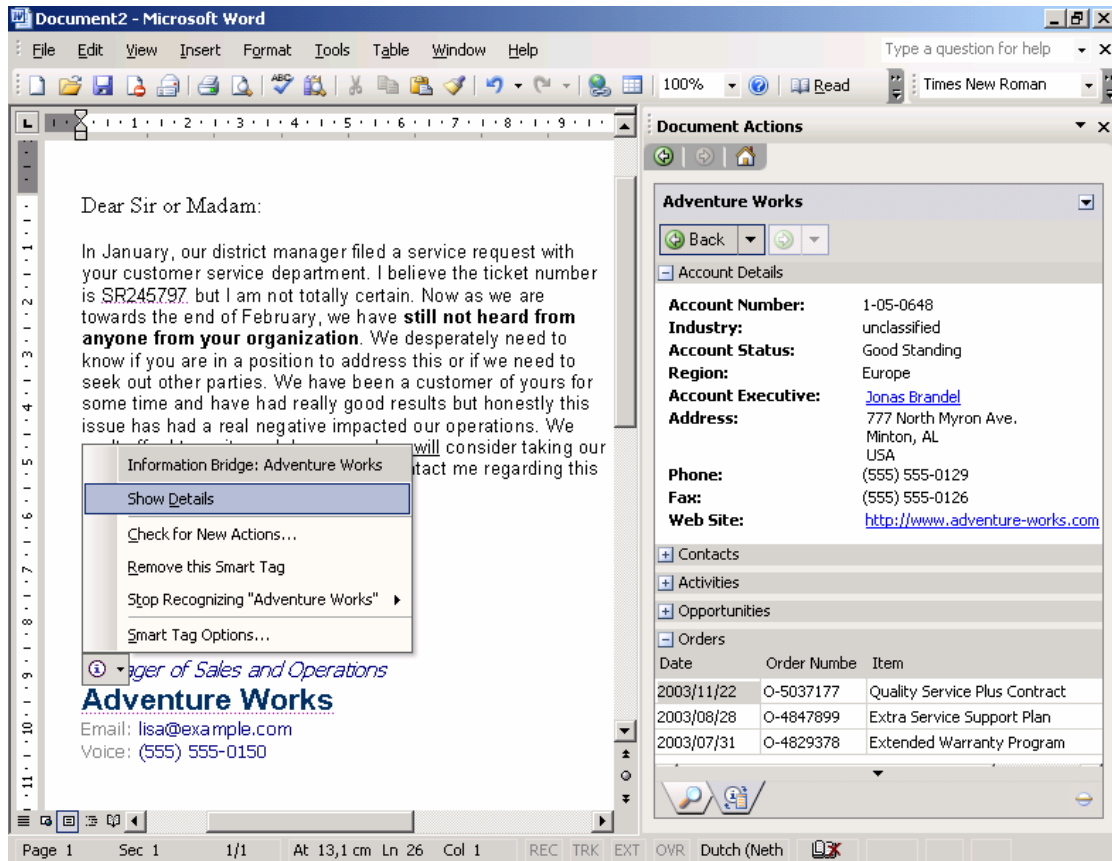


Figure 1: A typical business scenario for the Information Bridge Framework: by using a smart tag in a Word 2003 document, IBF connects to the Line-Of-Business application that provides the corresponding business data and services in a task pane.

Developing an IBF Application

From the information worker point of view a typical IBF application resembles any other Word, Excel or Outlook document that interacts with the task pane. But internally an IBF application consists of a number of components on the desktop, a metadata service provided by the IBF server and the web service facades for one or more Line-Of-Business applications (figure 2).

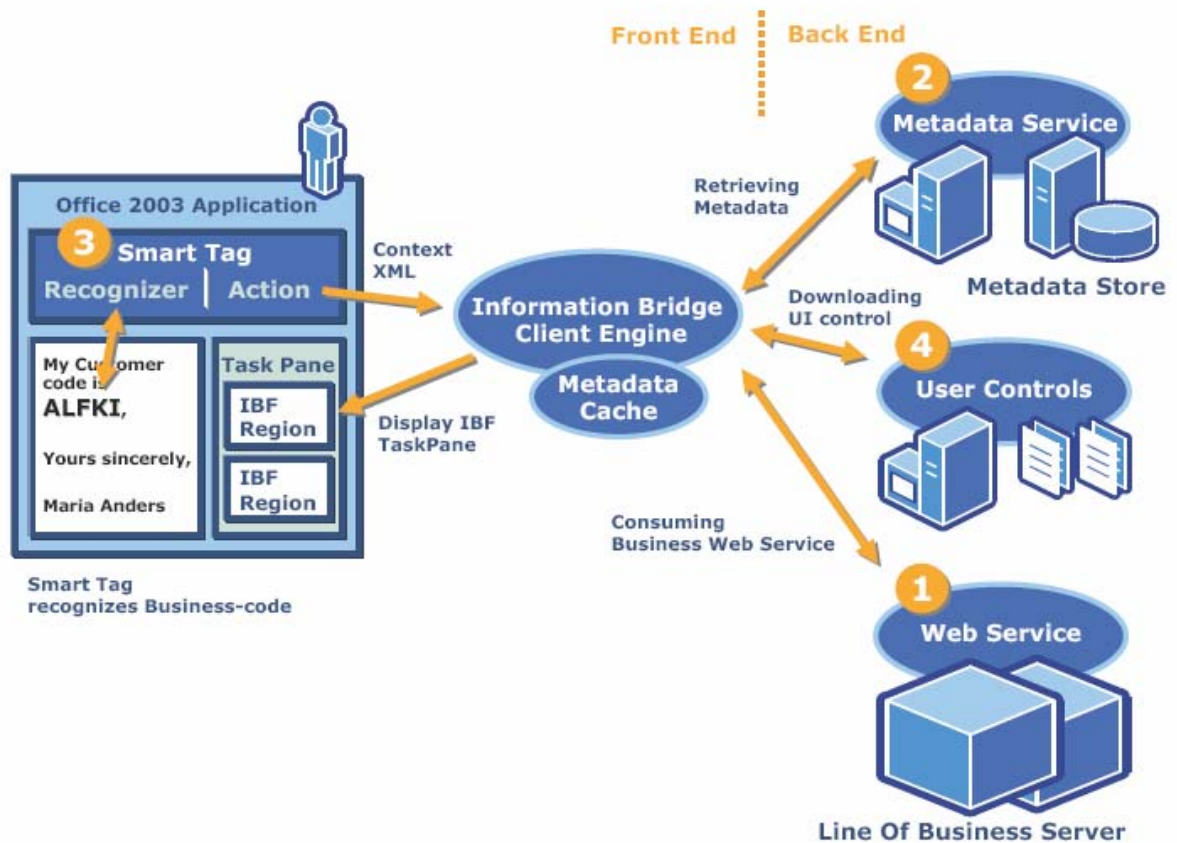


Figure 2: The parts of an IBF application: a LOB-web service, the Metadata service, the smart tag and the UI control.

On the desktop IBF requires Microsoft Office 2003 Professional, the Microsoft .NET Framework 1.1 and Windows 2000 or higher. The front-end parts run inside Microsoft Office 2003 applications and offer data which they exchange with web services on the server. The different parts are: the smart tag components, the user interface components for the task pane, and the Information Bridge client engine. This client engine fetches metadata from the IBF server, keeps this data in its cache, and offers the interpreted metadata to the Office environment by context sensitive navigation, menus, business objects and other information elements.

On the server side, the IBF metadata service runs on Windows Server 2003. The actual metadata is persisted in a so called metadata store, managed by Microsoft SQL Server 2000. The metadata describes in a standardised way the views, actions, relations, business entities and other information elements of the IBF application.

Developing an IBF application is not easy in version 1.0: the IBF developer environment is available as an add-in for Visual Studio.NET, but not really as a rapid application development (RAD) tool. The number of designers and wizards is rather limited, which result in quite a lot of manual development and configuration work. The first IBF application which you'll develop will require a significant amount of training and planning.

By using version 1.0 we can distinguish five development steps to create an IBF application:

1. Create a web service in accordance with the IBF rules
2. Create the IBF meta data
3. Create the user interface that will be shown in the task pane

4. Define either the smart tags or the attached schema document
5. Install and test the IBF application

For more details concerning the development tasks, we refer to the document "IBF in 5 steps, U2U tutor on the Information Bridge Framework" (<http://www.u2u.net/ibf>). To keep things convenient and foremost relevant, the IBF technology is showed by using the well-known Northwind database (Microsoft SQL Server 2000).

The parts of an IBF application

Let's take a look at the straightforward business case in which an information worker places the cursor on a customer code in a Word document and retrieves the corresponding customer information in the task pane. The information worker points out the customer code 'ALFKI', so a smart tag is displayed. When he selects the menu 'Show Details' the task pane will show company information about Alfreds Futterkiste (figure 3).

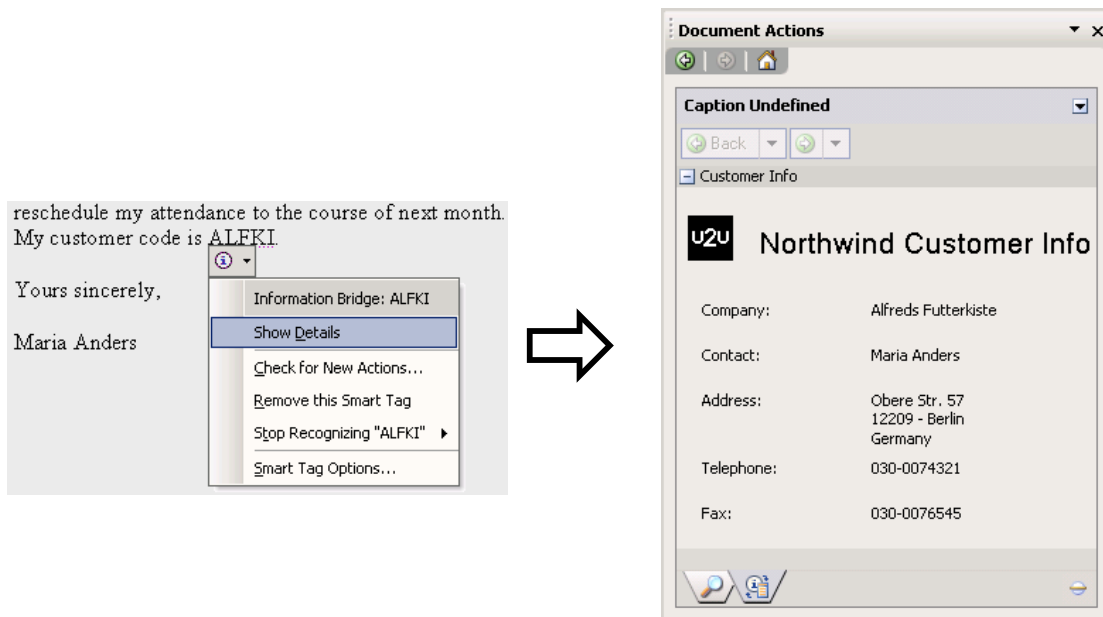


Figure 3: The business case which will be discussed in this article: the customer code ALFKI is recognized by the smart tag, resulting in the company information being displayed in the task pane.

So what happens behind the scenes? Which information is transferred by which means? Let's take a look at the different components of the IBF application and describe the information flow, starting from the recognition of the customer code as a smart tag, till the company information is displayed in the task pane.

Part 1: The web service in accordance with IBF

IBF customers retrieve their business data by using web services. These web services need to build in accordance with the Information Bridge Framework guidelines. This means for example that their definitions must correspond with schema's in the IBF metadata store.

In our business case we use the Customer web service with the GetCustomerInfo web method. This method takes the customer code as input value; the output value is the customer information. The IBF guidelines force us to encapsulate the customer code as a reference type – in our case the CustomerReference class. Besides that the customer information must be encapsulated in a view type – the CustomerView class (figure 5).

```
[WebService(Namespace="urn:schemas-u2u-net/Customers" )]
public class Customers : System.Web.Services.WebService
{
    [WebMethod(Description="Retrieve info for a specific customer")]
    public CustomerView GetCustomerInfo(CustomerReference customerRef)
    {
        ...
    }
}
```

Figure 4: Web services need to comply with the IBF guidelines.

```
[XmlRoot("CustomerReference", Namespace="urn:schemas-u2u-net/Customers" )]
public class CustomerReference
{
    [XmlAttribute]
    public string CustomerID;
}
```

Figure 5: The reference type (input) in accordance with IBF.

```
[XmlRoot("CustomerView", Namespace="urn:schemas-u2u-net/Customers" )]
public class CustomerView
{
    [XmlAttribute]
    public string CustomerID;
    [XmlElement]
    public string CompanyName;
    [XmlElement]
    public string Address;
    ...
}
```

Figure 6: The view type (output) in accordance with IBF.

Part 2: The metadata store on the IBF server

The metadata store in an IBF application contains all information concerning the business data that is needed to respond to a request that is coming from an IBF client, for example a smart tag. The metadata store describes for example the structure of the business data (schemas), where the business data resides (viewlocators), what can happen with the business data (actions) and how the business data must be displayed (transformations).

The metadata store is automatically installed as a Microsoft SQL Server database during the setup of IBF. The metadata store can be managed inside Visual Studio.NET by using the IBF Project template and an add-in (Metadata Explorer, figure 7). As well the IBF template as the Metadata Explorer is automatically installed during the setup of IBF.

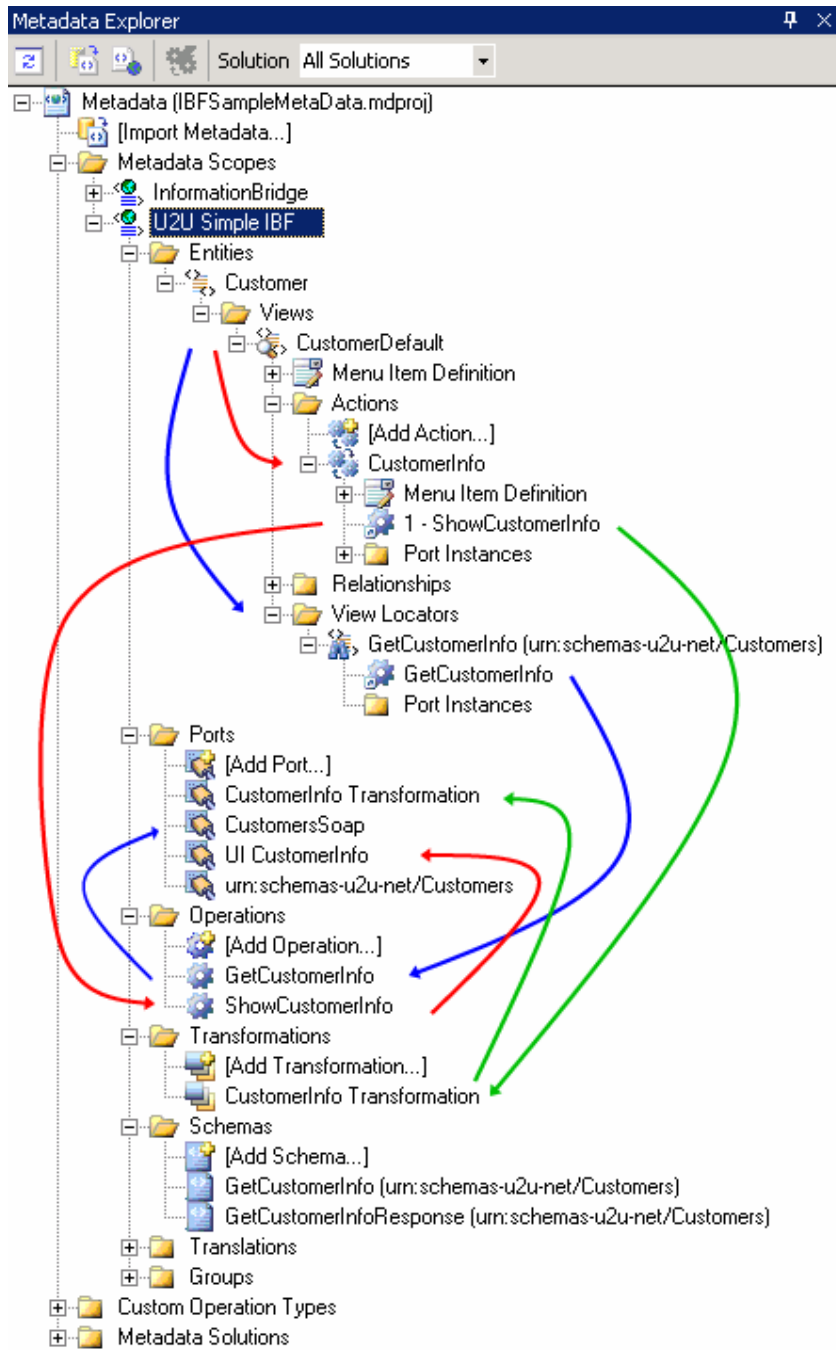


Figure 7: The Metadata Explorer

Metadata can be transferred between the metadata store and the metadata project through the metadata web service (for example <http://U2Ustore:8082/IBFWriteService.asmx>) as shown on figure 8.

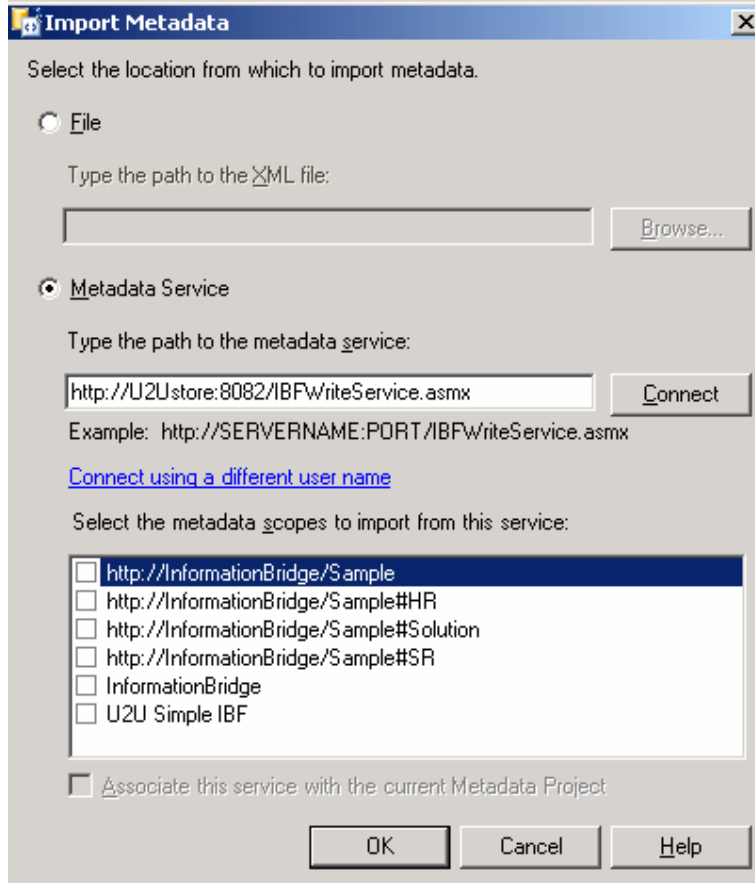


Figure 8: Importing metadata goes from the metadata store to the metadata project; publishing of metadata goes to the store.

Adding metadata to the project can be done by different means: manual editing the xml file, through wizards (Actions, Ports, Operations, Schemas ...) or by importing of xml files.

The Metadata Explorer allows managing important metadata elements. These elements are:

- Ports – the ports to the web service abstract from the SOAP details, and the ports to the XSD schema files from the input and output types.
- Schemas – the schemas of the IBF web service, the request and response formats are separated.
- Operations – the methods that can be called by IBF.
- Entities – the data entities and their views that will be used by IBF.

Part 3: The smart tag component or the “attached schema document”

Information workers can work with business data as text in an Office document. A smart tag is an object that is created at runtime and is embedded in the Office document. The smart tag technology enhances the Office document so it becomes a potential part of an IBF application. So if the information workers create a new document or alter an existing one, the smart tag allows us to create the link from the Office document to the IBF application. A smart tag is defined by a smart tag component which typically is created as a .NET class library project.

Part 4: The IBF user interface

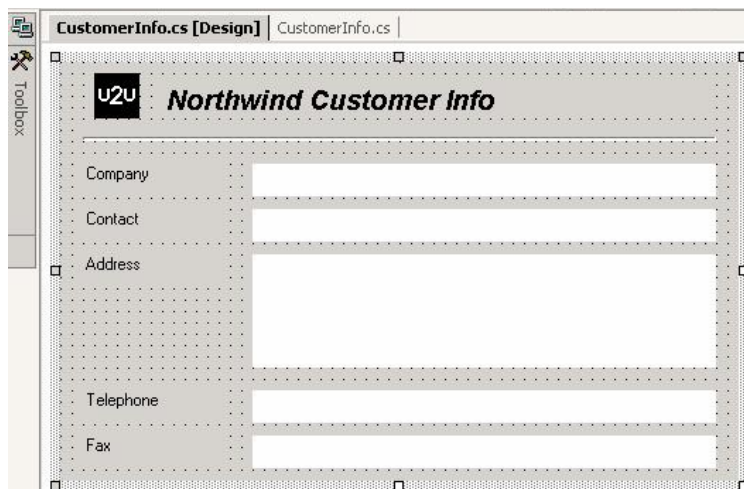


Figure 9: The IBF user interface is created as a Windows user control in .NET.

An essential part of the IBF application is the user interface in the Office environment. In Word and Excel the information worker uses the task pane, in Outlook this task pane is shown as a top level window. The user interface is build in our sample application, is built as a Windows user control in .NET, but IBF supports also plain HTML, built-in IBF controls (such as the list control) and the transformation language XSL for the definition of the interface inside the task pane.

IBF in action

Let's take a look at the IBF functionality from the moment a term is recognized as a smart tag in Word, until the displaying of the company information in the task pane.

Step 1: The company code is recognized

Let's start with the information worker that types a text in Word 2003 in which the phrase 'ALFKI' is present. ALFKI gets recognized by the smart tag engine of Office as a customer code. Behind the scenes the text got scanned by the SmartTagRecognizer (figure 11) of our smart tag. The smart tag (as described above in part 3) is working in Office and compares each word in the text with, in our case, company codes in an xml file named SmartTagTerms.xml. Our smart tag component recognizes the ALFKI as a company code it's in this xml file (figure 10).

```
<Terms>
  <Customer>ALFKI</Customer>
  <Customer>Anton</Customer>
</Terms>
```

Figure 10: SmartTagTerms.xml contains the ALFKI customer code.

The task of the smart tag is to build an xml string, conform to the schema defined in the IBF metadata store, which contains the context information that will be processed by the IBF client engine subsequently. The recognizer prepares this after the company code is recognized (figure 11). The smart tag underlines the term ALFKI in the text so the information worker can activate the smart tag with the cursor (figure 3).

```
public class SmartTagRecognizer : ...
{
    void Recognizer2(...)
    {
        ...
        switch (node.Name)
        {
            ...
            case "Customer":
                contextString = CustomerContextXml;
                break;
        }
        ...
    }
}
```

Figure 11: The smart tag recognizer.

Step 2: The request is processed by the IBF client engine

If the information worker clicks on "Show Details" in the context menu of the smart tag, the Execute method of the Action object in the smart tag sends an xml message (figure 12) to the IBF client engine. This engine accepts the message and processes it based on the metadata that was fetched from the metadata store.

```
<?xml version="1.0"?>
<ContextInformation xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  MetadataScopeName="U2U Simple IBF"
  EntityName="Customer" ViewName="CustomerDefault"
  xmlns="http://schemas.microsoft.com/
    InformationBridge/2004/ContextInformation">
  <Reference>
    <GetCustomerInfo xmlns="urn:schemas-u2u-net/Customers">
      <CustomerReference CustomerID="ALFKI"
        iwb:MetadataScopeName="U2U Simple IBF"
        xmlns:iwb="http://schemas.microsoft.com/InformationBridge/2004"
        iwb:EntityName="Customer" iwb:ViewName="CustomerDefault" />
    </GetCustomerInfo>
  </Reference>
</ContextInformation>
```

Figure 12: The xml message that is sent by the IBF client engine to the IBF server.

The format of this xml message is determined by the metadata on the IBF server, to be precise the GetCustomerInfo schema, which refers to the port urn:schemas-u2u-net/Customers.

This xml message contains a GetCustomerInfo request, with the CustomerReference that contains the CustomerID as an argument. The request refers to the Customer entity and the CustomerDefault view. In fact the xml message requests to display the Customer information of ALFKI, according to the CustomerDefault view.

Step 3: Fetching the required metadata by the client engine

The xml context message is answered by the IBF client engine, in which understandably the CustomerDefault view in the metadata starts working, of course in the perspective of the Information Bridge Framework. The answer provided by the IBF client engine is constructed based on the

metadata that's fetched and cached by communicating with the metadata web services of the IBF server component. The metadata in our sample consists of three parts: the url of the business web service, the url of the .NET assembly that contains the user control and the name of the user control class. With the help of the Metadata Explorer (figure 7) we can distinguish the different parts:

Part 1: the url of the business web service

The CustomerDefault view goes to the ViewLocators (blue arrows in figure 7) and finds over there the operation GetCustomerInfo that has to be executed to fetch the Customer data. The operation GetCustomerInfo points to the CustomerSoap port which is the first part of the response, by using a property. In our sample this is the location of the business web service:
<http://www.u2u.net/CustomerServices/Customer.asmx>.

Part 2: the url of the .NET assembly

The CustomerDefault view goes to the Actions (red arrows in figure 7) and finds the CustomerInfo action. This action points to operation ShowCustomerInfo. In the Operations section, ShowCustomerInfo points to the UI CustomerInfo port. In the Ports section, the UI CustomerInfo port points to the url of the Windows control DLL, in this sample: IBFNorthwindUserControls.dll.

Part 3: the name of the usercontrol class

One of the properties of the ShowCustomerInfo operation in the Actions section, the TransformationInstances property, points to (green arrows in figure 7) the CustomerInfo transformation. In the Ports section we can find the Data property which contains an XSL document (figure 13).

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <msibf:Region
      xmlns:msibf="http://schemas.microsoft.com/InformationBridge/2004"
      Enabled="true">
      <msibf:RegionProperties RegionName="RegionCustomerInfo"
        Caption="Customer Info" Description="Customer Info"
        TypeName="IBFNorthwindUserControls.CustomerInfo"
        ShowAs="ExpandedRegion">
        <xsl:copy-of select="/" />
      </msibf:RegionProperties>
    </msibf:Region>
  </xsl:template>
</xsl:stylesheet>
```

Figure 13: The XSL property that points to the User Interface.

The XSL contains a link to the usercontrol class named "IBFNorthwindUserControls.CustomerInfo", it is used and shown by in the region that's part of the IBF user interface (figure 9).

The final response of the IBF metadata service consists of:

- "http://www.u2u.net/CustomerServices/Customer.asmx" the url of the business webservice
- "IBFNorthwindUserControls.dll" the url of the .NET assembly
- "IBFNorthwindUserControls.CustomerInfo" name of the user control class

Step 4: the answer is processed by the IBF client engine

The IBF client engine makes the actual call to the business web service and shows the result in the IBF task pane. The user control receives the business data through the IRegion.Data property; this data contained in a XmlNode in accordance with the GetCustomerInfoResponse schema. The Data property of the user control takes care of the actual displaying of the customer information in the IBF region (figure 14).

```
public class CustomerInfo : UserControl, IRegion
{
    ...
    public XmlNode Data
    {
        set
        {
            labelCompany.Text = value.FirstChild.ChildNodes[0].InnerText;
            labelAddress.Text = value.FirstChild.ChildNodes[2].InnerText +
            ...
        }
    }
}
```

Figure 14 The Data property of the user control takes care of the actual displaying of the customer information in the UI controls.

The desired result is obtained for the business case discussed in this article, but in a business case where the data needs to be managed or where business activities must be started, there's analogous work to do.

References

MSDN on the Information Bridge Framework:

<http://msdn.microsoft.com/office/understanding/ibframework/default.aspx>

"IBF in 5 steps, U2U tutor on the Information Bridge Framework"

<http://www.u2u.net/ibf>

About the authors

Wim Uyttersprot (wim@u2u.be) and Patrick Tisseghem (patrick@u2u.be) are managing partners of U2U, Gert Servranckx (gert@u2u.be) and Jan Tielens (jan@u2u.be) work at U2U as .NET trainers. U2U is a certified .NET Trainings Center, located in Brussels, specialized in .NET development. U2U takes care of a number of workshops for Microsoft EMEA (www.u2u.net).