At a glance:

- Gathering and processing data
- Separating presentation logic from information management logic
- Using Microsoft Office 2007 technologies to build a data-gathering solution

A smart approach to gathering data in the enterprise

Keith Deshaies

Enterprises gather vast amounts of information using a variety of methods. The data arrives via e-mail, surveys, Web forms and other data collection mechanisms. Data, usually, is a good thing. However, managing the array of data collection tools and all the disparate information is difficult. Reliable integration and secure sharing of data are constant challenges for IT organisations. Standards and service-oriented architectures are evolving, making it easier for IT professionals to make data more accessible, more securely. But while you have the tools and technologies you need to build an efficient enterprise architecture, it is far too common to get caught in a net of proprietary interfaces – and this leads to isolated solutions.

Take the technologies available in the 2007 Microsoft Office system as an example. You can quickly create a departmental survey based on Windows SharePoint® Services 3.0, but whether this is a standard solution depends on your organisation. If your company uses ASP.NET and SharePoint as the platform for Web-based collaboration and data integration, then this survey does provide a standard solution. But if your environment is like the one I work in, SharePoint is just one platform among many.

Granted, SharePoint provides many op-

tions for integrating with systems from IBM, HP, Siebel, and so on. That's good news for power users who want to create ad hoc solutions and still have the resulting data flow into a variety of back-end systems. However, if you're a solution architect, there is an even better way to go – InfoPath® 2007.

With InfoPath 2007, which is part of the 2007 Office system, you can decouple the presentation logic of your solutions from the information management logic hosted on your servers. With XML-based InfoPath technology, you can build enterprise-ready data-gathering solutions. And, for the most part, InfoPath form designers do not need detailed knowledge of XML, Web services, solution architectures, ASP.NET or the SharePoint object model.

In this article, I will discuss how you can build flexible data-gathering solutions using InfoPath 2007, Office SharePoint Server 2007 and Forms Services. And I'll show you how XML enables you to separate the presentation logic from the business logic in a multitier, enterprise architecture.

Note that when I refer to "data gathering" I am referring to the process of collecting information from human sources. There are, of course, other ways to gather data, such as crawling data sources, but those automated methods are beyond the scope of this article.

Acquiring and handling data

Data aquisition requirements can be complicated, but these processes have some things in common. By addressing these similarities in centralised modules while handling unique requirements in decentralised components, you can limit redundant efforts, maintenance overhead and the total cost of ownership.

For example, compliance regulations for public companies result in business requirements that in turn translate into company-wide information management policies. These policies affect data gathering solutions across departmental boundaries and often lead to duplicated efforts within individual departments – for instance, rules around the collection of personally identifiable information gathered by an HR department (handling employee info) and a customer- service department (handling customer info). Even business processes between individual departments that are similar but unrelated provide opportunities for unifying information management solutions.

Figure 1 shows an example of a typical business process. An employee who wants to trade an assignment with a colleague must first obtain an agreement from the colleague, then approval from a manager or coordinator of the assignment schedule, and finally from the supervisor. This could involve employees trading work shifts, for example. Though these exchanges occur in different departments and may rely on different forms, the workflows and information management logic may be shared among the various department solutions.

Of course, consolidating redundant components is a huge task. Driving organisational change across a company is not easy, but with the Office system technologies you can build a solid foundation to facilitate these changes. InfoPath 2007 enables individual departments to create forms applications that integrate with centralised, standardised, information management systems. SharePoint 2007, meanwhile, enables IT departments to delegate administrative control over site collections, sites and document libraries to individual departments and teams. As a result, teams can build and deploy their own solutions with minimal involvement from IT, while the IT department remains in control of all the shared services and components,

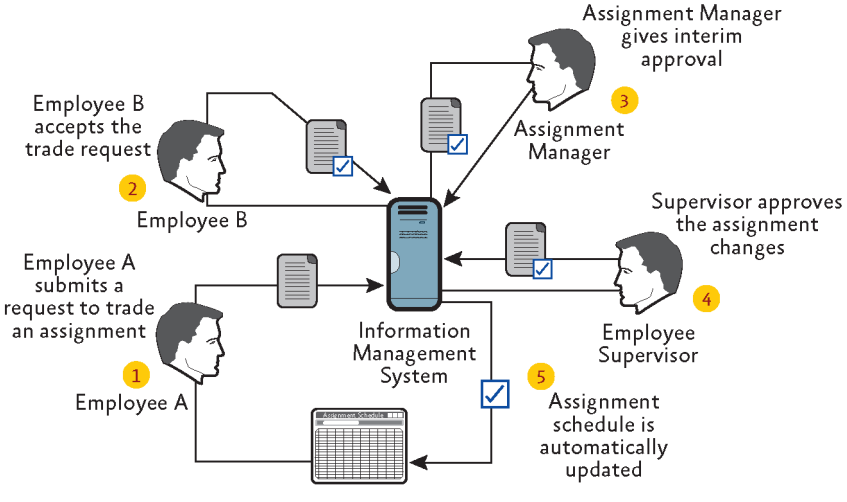

Figure 1 **Data-gathering process that may be shared among departments**

such as workflows, information-management policies and backup procedures.

Centralising your data-gathering efforts

Enterprises often give teams departmental application servers to accommodate individual information management needs. The IT department is merely responsible for keeping the hardware and operating system running, while the individual departments take care of all aspects of their solutions. There is little coordination between departments, and information sharing is difficult.

Technical challenges to centralising data-gathering efforts revolve mainly around the security, performance, maintenance and support of custom components hosted in a shared environment. For instance, the effects of a malfunctioning component are isolated if individual solutions are hosted on departmental application servers. In a shared environment, however, a malfunctioning component can affect business processes on a much larger scale. It follows that the IT department must establish strict policies regarding the deployment and maintenance of custom components on centralised systems.

Hosting departmental SharePoint solutions on a central server farm requires that you deploy and maintain all the custom components of these departmental solutions on the central application servers. One solution might rely on custom field types to extend the solution's UI with dropdown lists populated from back-end Web services. Another solution might rely on Web Parts for the same purpose while yet another uses custom workflows – all of which are written in managed code and deployed as Microsoft .NET Framework assemblies.

Moving even a relatively small number of SharePoint solutions to a central application server farm can lead to difficult configuration and support issues. If assemblies must be deployed in the Global Assembly Cache (GAC), security becomes an issue because these assemblies run with full trust. Poorly programmed components might open the system to SQL injection, cross-site scripting or denial-of-service attacks. You need to ensure that the components can sustain the typical workload as well as peak demand and long-running operations. You need to ensure that the components don't block other processes, handling events synchronously, and that the components perform reliable input validation – so users cannot insert SQL statements or scripts into columns used to update a database or remote Web system.

In short, the goal is to emphasise secure and scalable server configuration based on standard product features. By relying on reusable, thoroughly tested solutions, you can avoid the trap of creating numerous custom components. It makes sense to keep the front end decentralised and the back end centralised. The key is to integrate the components in a loosely coupled way that promotes reuse of existing solutions.

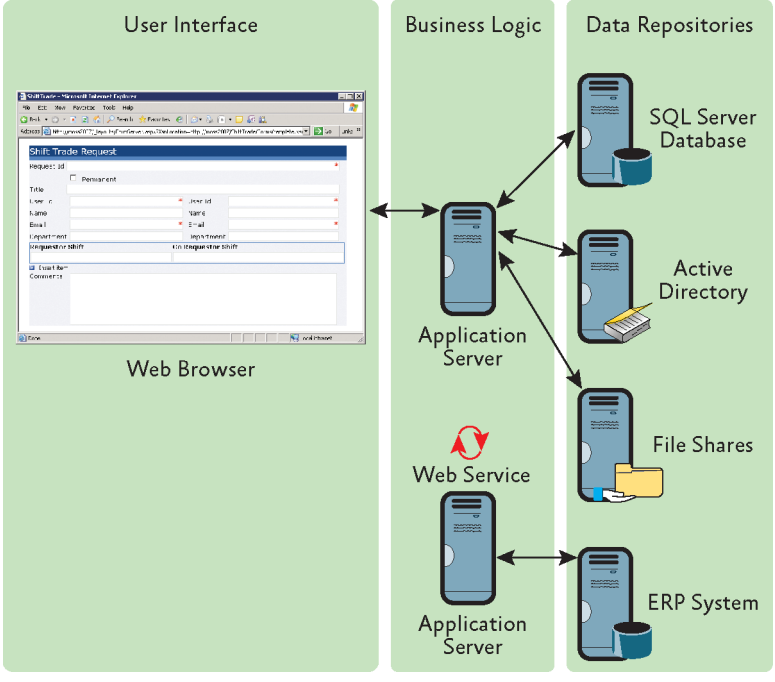

Figure 2 **A typical enterprise solution built on a three-tier architecture**

Splitting the business logic

So how do you build flexible data-gathering solutions that can be configured on your servers? The best strategy is to separate the solution architecture into individual tiers as shown in **Figure 2**: data storage, business logic, and presentation or UI. These days, the UI is typically browser-based while the business logic resides on Web application servers. These, in turn, access databases and non-relational data repositories.

Business logic often includes transaction logic to ensure that transactions are applied

atomically across database management systems. Business logic may also integrate multiple middle-tier services through HTTP, message queuing, RPCs, and so on. The overall solution architecture, however, remains essentially a three-tier model.

What **Figure 2** does not illustrate is the complexity of the business logic in an enterprise environment. It looks as if the application server in this figure is merely focusing on rendering a browser-based form and handling the submitted data, but that representation does not take into account workflows, compliance or information management requirements. To address these requirements, you need to split the business logic in two – the presentation logic and the information management logic. This allows you to mix and match the middle-tier components as needed without rebuilding components from scratch for each solution.

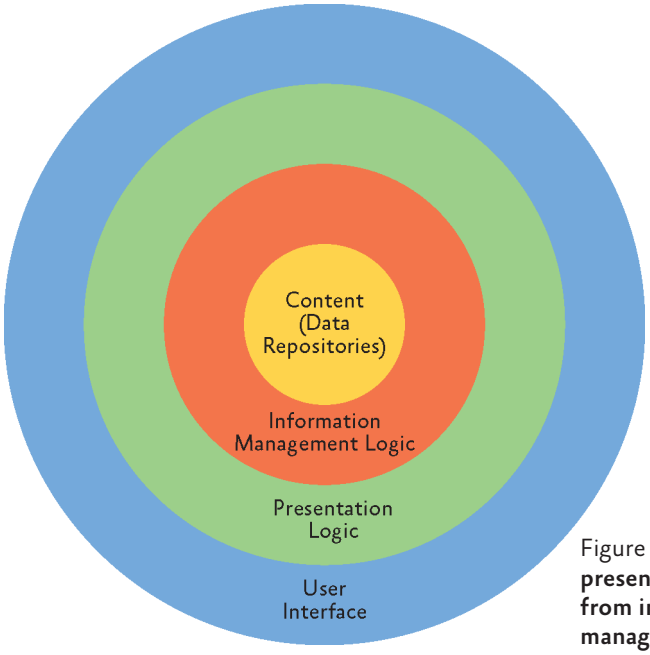

Figure 3 **Separating presentation logic from information management logic**

Figure 3 shows this architecture. At the core is the content or data, surrounded by the information management logic, which governs the content throughout its lifecycle. The presentation logic interfaces with information management logic to provide access to the data via the user interface.

Gathering and processing XML

In service-oriented application (SOA) environments, XML is the dominant standard used to define and share data and data structures among components. And XML, therefore, is a good choice for interfacing between presentation and information management components.

Communication must move in two ways: you'll need to translate the XML into a browser-readable document, as well as an XML document generated by the form. Until recently, building XML-based forms applications required extensive programming skills. This was especially true when the resulting XML data had to adhere to an industry schema to facilitate interorganisational information exchange.

InfoPath 2007 makes XML-based forms development much easier. A strong grasp of XML details is certainly helpful, but forms designers and power users need not be XML wizards to build XML-based forms apps. The forms designer simply imports an XML document or XML schema into InfoPath and then maps the individual attribute and element nodes from that data source to the fields in the form. A forms designer can also start with a Web service or a SQL Server® database or from a blank template and build a form from scratch while InfoPath automatically creates the underlying schema and data bindings in the background.

Standardising forms using InfoPath and XML schemas has several advantages. If you already have a well-defined XML schema, forms designers and developers of workflows and information management components can create solutions against the same data structures. If a forms designer starts from scratch, developers must wait for the form to be finished in order to see how it affects the underlying data structures. And once the data structures are defined, future solutions, such as new form templates, can reuse existing workflows and information management components if they rely on the same data structures. And future workflows and information management components can work together with existing forms and data. If you build your data-gathering solutions based on established industry schemas, the results become even more flexible. In fact, these solutions will be compatible with solutions built by other companies that use the same schemas.

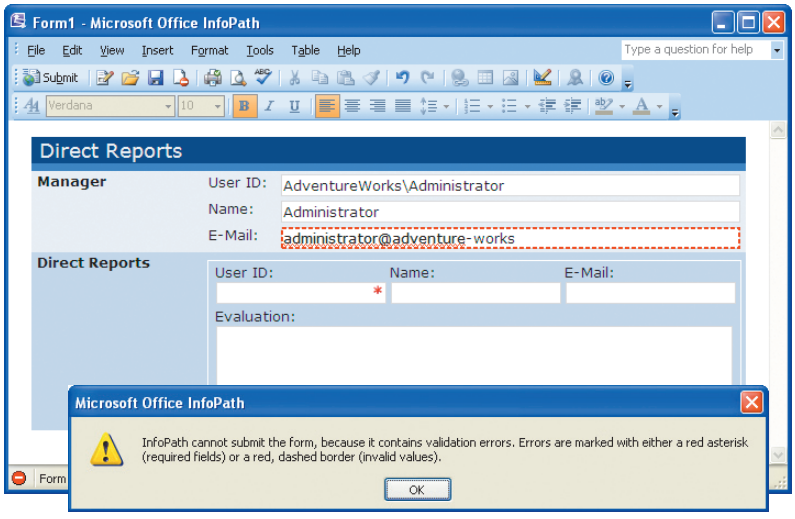

Figure 4 **Validation errors prevent the user from submitting a form**

I created a simple DirectReports schema that associates employees with managers. Managers can use the resulting form to evaluate their direct reports. You can find the schema, the form and a readme.htm with instructions to recreate the form in the Direct Reports folder in the download available at technetmagazine.com/code08.aspx. The form is basic, but it illustrates the general concept.

A very important point here: I did not create any validation logic in InfoPath, yet InfoPath still requires that user ID and e-mail addresses be entered in a specific format (domain\account and recipient@domain.tld). Otherwise, the resulting XML document is not valid. This is because the XML schema defines these formats. You can save the form with invalid data but you can't submit it, as shown in **Figure 4**. (I've added a dummy submit rule to the form so you can test this without actually submitting data to any location.) InfoPath 2007 validation automatically ensures that the form is filled out completely and without these sorts of errors.

The XML schema serves as the binding contract between the presentation logic and the information management logic. InfoPath locks the schema so the forms designer can't intentionally change the data structures. This is important because changing an established XML schema can potentially break existing enterprise solutions, such as workflow modules that you intend to use in combination with the new forms template.

InfoPath provides an abundance of features for building advanced presentation logic into forms applications. You can consume data from XML files, Web services, SharePoint libraries and lists, databases and so on to pull in meaningful default values. You can change field values based on user selection through rules, include validation logic, add managed code for the most advanced customisation requirements, and use Forms Service to make the form template accessible over the Web. In any case, the data from the form eventually reaches the information management logic as an XML document that conforms to a schema definition.

Working with XML or metadata

You might wonder whether you should apply information management logic directly to the submitted XML document or instead use a parser that extracts the required information into metadata. SharePoint supports both approaches. Developers are accustomed to working directly with XML documents, but metadata offers more flexibility.

To demonstrate this, I created a Web service that parses an XML document passed in from the Direct Reports form shown in **Figure 4**. (The source code, setup files and a readme.htm with step-by-step instructions are in the XMLParsingWebService folder in the accompanying download.) The Web service reads the manager's user ID from the XML document, splits the user ID into domain and user name parts, creates a message based on these parts, and raises a generic exception to return and display the processed information in the form of a pseudo-error message in the InfoPath form. This is an easy way to pop up a dialog box in InfoPath after data has been submitted. The Web service works great, but if you change the underlying data source (for example, if you rename the OrgPerson element as Manager in DirectReports.xsd and update the InfoPath form with the new schema as outlined in the readme.htm file) the Web service fails. But this shouldn't be a surprise. The XML document is now different and the old XPath expression to access the user-id element is invalid. The OrgPerson and Manager schemas are almost identical, the InfoPath forms are identical and the desired processing results are the same, but

despite the differences being minimal, you still need to deploy and maintain duplicate Web services.

This is not a good approach if you want to minimise the footprint of custom code on your application servers. In contrast, mapping the XML nodes to metadata fields and performing the processing based on the metadata allows you to use the same workflows and information management logic for similar data structures even though the XML schemas are different. You just need to make sure that the child element maps to the correct metadata field and that the data format meets the processing requirements – then you can reuse the existing code.

Mapping XML nodes to metadata fields is similar to binding XML nodes to UI controls in an InfoPath form, as shown in **Figure 5**. In SharePoint, metadata fields correspond to columns defined at the site or list level and referenced in content type definitions. Content types define the characteristics of content items, such as metadata fields, workflows and forms. To keep the metadata fields of a content type synchronised with the corresponding nodes in the associated XML document, SharePoint relies on a built-in XML parser that performs property promotion and demotion. During property promotion, the XML parser extracts node values from an XML document into corresponding columns on the document library. Property demotion refers to the reverse process in which column values are taken from the document library and written back into the document. The most important point is that the XML parser keeps the metadata fields and mapped XML nodes synchronised.

When you use the SharePoint object model, your Web Parts, workflows and information management logic can work with metadata fields as well as the underlying XML documents. Changing the value of a mapped metadata field changes the node value in the XML document, and vice versa. Yet, working directly with the XML document tightly couples the business logic with the XML schema. Mapped metadata fields, on the other hand, increase the reusability of code. Obviously, you need to decide which approach is right for your environment, but for the most part SharePoint solutions that rely on metadata fields provide more flexibility and more opportunity for code reuse.

To illustrate how SharePoint associates XML nodes with metadata fields, I included a SharePoint feature in the companion material to provision a custom document library (see the OrgPersonContentType.xml file in the OrgPersonLib folder in the accompanying download). The OrgPerson content type references four fields: UserID, FullName, EMail and Direct_x0020_Reports. FullName and EMail are built-in fields. UserID and Direct_x0020_Reports are custom fields defined in OrgPersonSiteColumns.xml.

The field definitions are fairly straightforward. It is possible to bind fields to XML nodes directly in the field definitions, yet it is

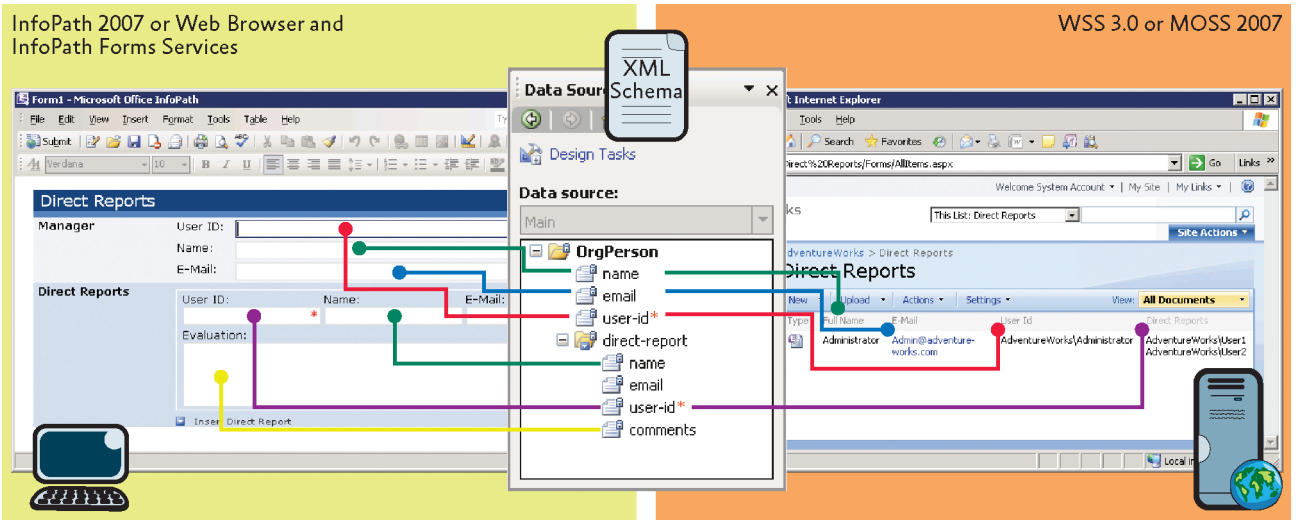

Figure 5 **XML Schema mappings between InfoPath and SharePoint**

Figure 6 **Metadata field mapped to an XPath expression**

```
<FieldRef ID="{D30074C8-2F88-4B38-B44E-C6A4E919120B}"
  Name="Direct_x0020_Reports" ReadOnly="TRUE"
  Node="/OrgPerson/direct-report/user-id"
  Aggregation="merge" />
```

also possible to overwrite this information in the content types. I decided to use the latter technique because this let me use the custom fields in content types not related to XML documents as well as in content types that rely on different XML structures. The OrgPerson content type binds the metadata fields to XML nodes that match in their arrangement the OrgPerson schema I discussed earlier. There is also an AdditionalContentTypes.xml file that defines more content types that bind the same metadata fields to different XML nodes. You can see the differences if you look at the XPath expressions in the Node attributes.

Document libraries of the OrgPersonLib type can store different types of XML documents while the node values are exposed through the same library columns. This simple mapping technique also lets you reuse workflows and information management logic since the four content types (OrgPerson, Manager, Supervisor and User) reference a common set of metadata fields.

Figure 6 shows you the FieldRef element from the OrgPerson content type for Direct_x0020_Reports, which maps the field to the user-id nodes of direct reports according to the XPath expression, /OrgPerson/direct-report/user-id. Since the XML document can include multiple direct report entries, it is important that you specify an Aggregation attribute. This defines how the XML parser will handle the returned collection of values. If you omit this attribute, the XML parser extracts only the first node value. Supported aggregation values are sum, count, average, min, max, merge, plain text, first, and last.

All of the sample content types use the standard upload.aspx page as the DocumentTemplate so that you can upload XML files into the document library when you click on the New button in the SharePoint UI. As long as you upload files with an .xml file-name extension, SharePoint will automatically invoke the built-in XML parser (one exception is WordProcessingML files, for which SharePoint invokes a WordProcessingML parser). The XML parser examines the uploaded .xml file to determine the associated content type. This is so it can extract the node values from the locations specified in the field definitions and perform property promotion. (You can verify this process when you upload the OrgPerson.xml file included in the OrgPersonLib\XMLFiles folder.) The structure of this XML document matches the XPath expressions specified in the OrgPerson content type definition. Accordingly, SharePoint extracts the data from the .xml file, writes the data into the corresponding library columns and displays the data in the EditForm.aspx page so you can verify and update the document properties that are not marked as read-only. **Figure 7** shows the EditForm.aspx form with the data extracted from OrgPerson.xml.

If you change the User ID, Full Name or E-Mail value in EditForm.aspx, SharePoint performs property demotion to change the node values in the underlying XML document. However, SharePoint does not enforce XML Schema restrictions unless you implement the required logic into the form yourself.

SharePoint does not run the presentation logic of a forms application. For example, when you change the User ID, SharePoint does not validate that the new value conforms to NetBIOS conventions and does not automatically update the Full Name and E-Mail fields to match the new selection. Thus, you should mark the corresponding column in the content type definition as read-only if changing an individual field may cause inconsistencies. This forces the user to use the forms application, such as InfoPath,

Additional online resources

- **InfoPath 2007 Team Blog**
  blogs.msdn.com/infopath
- **W3C XML Schema Specifications and Development**
  w3.org/XML/Schema#dev
- **Introduction to XML Schemas**
  msdn2.microsoft.com/efc70bx3
- **Workflows in Office SharePoint Server 2007**
  msdn2.microsoft.com/ms549489
- **InfoPath 2007 Forms Services**
  msdn2.microsoft.com/ms540731

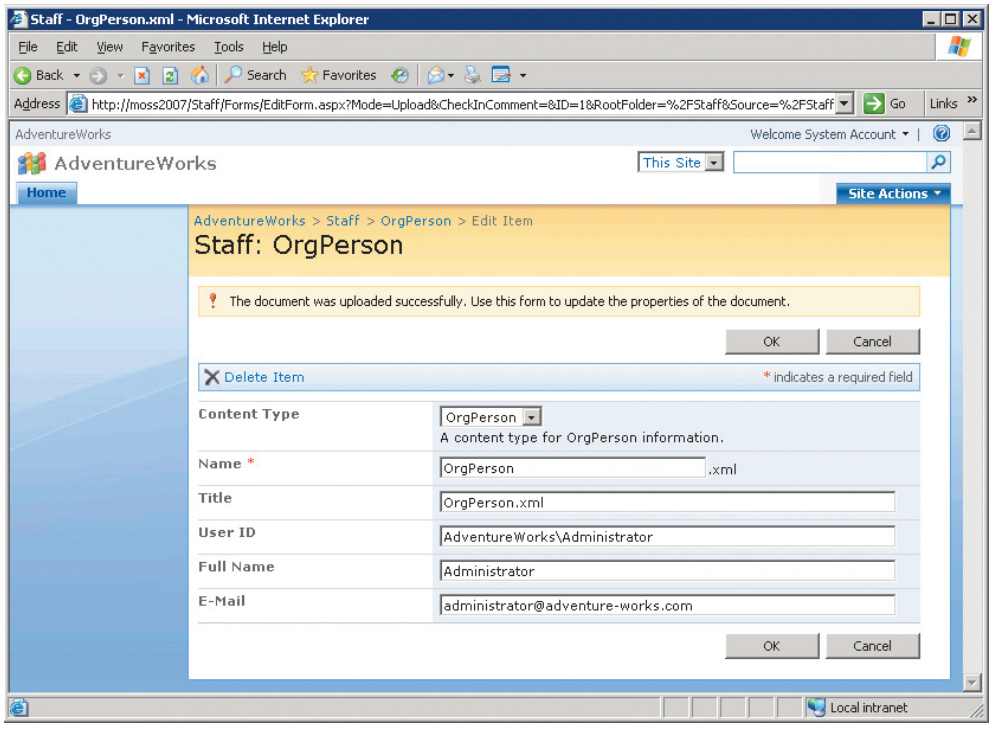

Figure 7 **EditForm.aspx form with extracted data**

to update the data. And the XML parser will promote any changes from the XML document to the corresponding metadata fields in SharePoint.

In the OrgPersonLib sample, you can upload any of the .xml files from the OrgPersonLib\XMLFiles folder. The .xml files use very different data structures, but SharePoint recognises the content types and promotes the correct node values into the site columns. This is because I used a processing instruction in the XML files to associate the XML documents with their corresponding content types. The OrgPerson.xml file, however, doesn't include this information, but this is not a problem. If SharePoint cannot associate an XML doc with a content type through a processing instruction or the document template, SharePoint uses the default content type. In the OrgPersonLib case, this happens to be the OrgPerson content type and therefore the XML doc is parsed correctly.

Figure 8 shows how you can associate an XML doc explicitly with a content type. The MicrosoftWindowsSharePointServices processing instruction defines the ContentTypeID as 0x010100668E393E4F0EFF4294DBD202D5D8321D. This happens to be the ID of the User content type, as defined in AdditionalContentTypes.xml.

The XML parser processes the MicrosoftWindowsSharePointServices processing instruction and writes the ContentTypeID value into the ContentType metadata field. All SharePoint content types share this metadata field because it is defined at the root level in the System content type. If you open the fieldswss.xml file on a SharePoint server (located in %CommonProgramFiles%\Microsoft Shared\Web Server Extensions\12\Template\Features\Fields folder) and search for MicrosoftWindowsSharePointServices, you can see how SharePoint associates the processing instruction with the ContentType field. The PITarget attribute points to MicrosoftWindowsSharePointServices (this is the

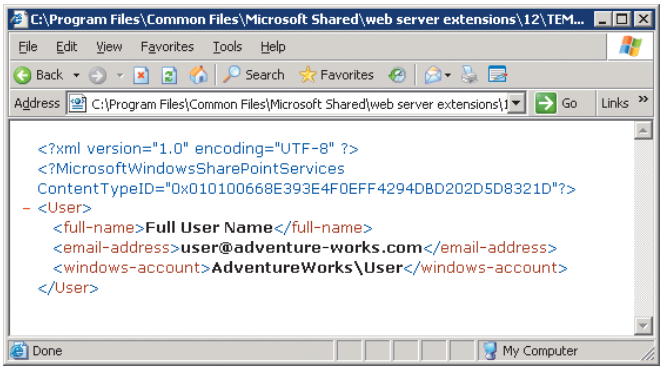

Figure 8 **Processing instructions and XML data of the User.xml sample file**

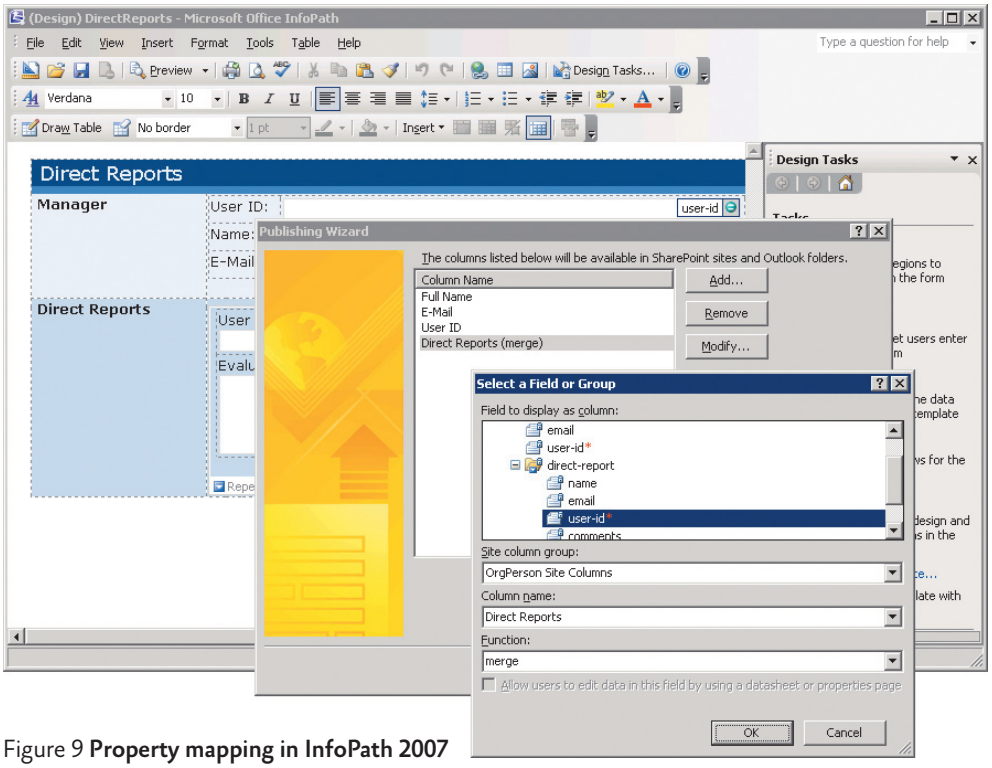

Figure 9 **Property mapping in InfoPath 2007**

processing instruction) and the PIAttribute points to ContentTypeID (which contains the ID of the User content type).

Content type associations in InfoPath

The technicalities of XML parsing and content type associations are intimidating for many forms designers, but InfoPath 2007 takes care of all the nitty-gritty details. The readme.htm file that accompanies the OrgPersonLib sample includes instructions to publish the Direct Reports form template in SharePoint and create a content type that binds yet again to the same metadata fields (UserID, FullName, EMail, and Direct_x0020_Reports). You can easily add the new content type to the OrgPersonLib document library in the SharePoint UI. But the new content type also points to the InfoPath form template as the document template to invoke the forms application when updating existing XML documents. **Figure 9** illustrates how the InfoPath Publishing Wizard simplifies property mapping between XML node values and SharePoint site columns. And, again, if you associate the node values with existing site columns, you can reuse existing SharePoint components.

Wrapping up

With technologies available in Office, enterprise architects can build data-gathering solutions that readily promote code reuse and information exchange. InfoPath 2007 allows departments to create solutions that can pull information from various sources, and the data can then be submitted to various systems, such as SharePoint. SharePoint also provides developers with a comprehensive set of Web services and interfaces to build workflows and information management components. By hosting these components on centralised SharePoint servers, departments then have the infrastructure they need to build their individual applications.

Meanwhile, individual departments can create their data-gathering solutions faster. Compliance regulations and other global business requirements can be addressed at a cross-departmental level, and the maintainability and reliability of the IT environment increases with the use of fewer custom components on application servers. ■

For more information about SharePoint please visit the SharePoint Tech Centre at: ***http://technet.microsoft.com/en-gb/office/sharepointserver***

Keith Deshaies *is a freelance technical writer and an IT analyst for a large telecommunications company. He specialises in Microsoft Office and SharePoint technologies and is a member of the Society for Technical Communications.*