Connecting e-Commerce to XML

Stephen Mohr, Omicron Consulting

Introduction

e-Commerce is an exercise in interconnection. If you truly want to realize the benefits of a frictionless, commercial Web, you need to connect your site to a host of vendors, suppliers, partners, and internal applications. A practice shooting to prominence in the field is the use of XML as the format for expressing and exchanging data. Applications and partners are connected through the exchange of XML documents describing the business task to be accomplished.

Simply deciding to use XML is not the end of the story. This presentation discusses the challenges of using XML and what techniques you can use to overcome them. We will discuss several methods for getting data out of common sources like databases, and into XML formatted documents first time.

A problem many people do not anticipate is the problem of data interoperability. XML by itself does not specify structures for e-commerce. It is more like a syntax and grammar which may be used by applications to create e-commerce languages. We will show you how to locate suitable applications of XML, termed vocabularies, that others are using, and how to translate to and from those vocabularies when you need to use your own specialized vocabulary.

Finally, you will receive a brief survey of commercially available e-commerce XML products and what they can do for your implementation.

Stephen Mohr is a software systems architect with Omicron Consulting, Philadelphia, USA. He has more than ten years' experience of working with a variety of platforms and component technologies. His research interests include distributed computing and artificial intelligence. Stephen holds BS and MS degrees in computer science from Rensselaer Polytechnic Institute.

Needs of B2B e-Commerce

Moving a business to the Web brings certain requirements that do not exist in businesses that are wholly manual, or at least those where the interconnections between work processes are manual. Whenever an employee receives a document and performs some processing – even if only to key it into an automated application – the business has the opportunity to apply human reason and common sense, thereby smoothing the way for applications. Such manual intervention adds time to the business cycle, however, so we want to find a way to avoid the need for human intervention at the points where applications interact. This is particularly true of business to business (B2B) operations.

The principal benefits of B2B e-commerce are, ideally, the reduced costs and shortened delivery times promised by automated systems. Outsourcing functions to suppliers and partners permits a business to concentrate on what it does best and reduce costs to a minimum. A secondary benefit is the ability to conduct business around the clock. Anything that adds manual intervention works against these benefits. Also, since one of the primary aims of B2B e-commerce is to reduce costs, you do not want your B2B interconnections and integration systems to introduce high costs of their own. It is imperative that e-commerce systems be implemented at minimum cost.

When you are exchanging data with a large number of external partners, such as suppliers, you quickly encounter the problem of data translation. Everyone tends to develop their own way of describing information and processes. Industry-standard formats are slowly taking hold, chiefly due to the promise of reduced development costs, but there are still many factors that drive businesses to create their own formats. In the physical world, we are used to seeing similar but different business forms for common functions like purchase orders. This is not a problem because human intervention provides a translation. In e-commerce, we must provide an automated and dynamic translation mechanism. Even if a majority of participants in a particular market agree to some set of universal data formats, we must still be prepared to translate to and from these formats and equivalent proprietary formats used by the minority.

In the recent past, automated e-commerce, chiefly **Electronic Data Interchange** (EDI), was limited to very large corporations. They were the only ones who could afford the expensive computers and proprietary networks required then. Today, though, we are accustomed to inexpensive computers and many people have access to the Internet. We expect to be able to conduct e-commerce with small and medium sized businesses. This will mean an explosion in the number of potential partners with whom you must be prepared to interoperate. Clearly, there is a need for most people to use standards for simple things that are not specific to their business, notably networking and user interfaces.

All these factors point to the Web. Secure IP-based networking over the public Internet promises commodity networking. Indeed, businesses can increasingly assume their partners will already have robust access to the Internet. In cases where manual intervention is required, such as a human-initiated purchase or authorization, web browser interfaces are an excellent means of providing applications to people without having to train them in the basics of using the application. While you may have to explain your particular business processes, you can expect users to understand clicking, links, and the behavior of HTML form elements. The combination of the public Internet and commodity web browsers spell the end of proprietary networks and applications, such as were required by older e-commerce systems like EDI.

B2B Technical Problems

Few companies will have the luxury of starting from the beginning and creating systems and data formats expressly for B2B e-commerce. The investment in legacy systems, data storage, and data structures and formats is simply too great to throw away for the average business.

Legacy applications have another name in most companies: the systems that have been running the business for years. They are trusted and dependable. Much has been invested in them and the day-to-day operation of the company relies on the continued and correct functioning of these applications. It is unwise and often impractical to modify them solely for the purpose of integrating with other applications. Interconnection between many of these systems is currently made through manual intervention. There will be great reluctance to open them up to applications hosted by outside partners. Your e-commerce system must take this reluctance into account. You must deal with the I/O already in place.

Data formats are another area in which companies resist change. There are the technical issues of understanding and converting large volumes of legacy data, not to mention the issue of modifying legacy applications to operate with a new format. There is also the cultural issue of changing formats. Often, the structure of a business message generated by a company's applications reflects that company's way of looking at its industry. My purchase order is your requisition. My way of listing business contacts is different from yours. Both are valid and related, but neither of us is going to change. Small companies, in particular, face this challenge. Any large customers are likely to dictate formats to them. Unless they want to rely on a single large customer, they must be prepared to accommodate multiple data formats. Dynamic translation is needed.

What is needed to answer these challenges is a simple and inexpensive way of expressing data. It must be well suited to the Internet and its protocols, and it must lend itself well to dynamic translation. As you might have guessed, XML is one such answer.

XML as a Common Representation

XML has a number of things going for it as a common format for data representation in Internet-enabled B2B e-commerce. First, it is entirely represented in text. Unlike binary formats, XML can be processed without change regardless of the receiving platform. Different countries and languages employ different character sets, but XML includes provisions for specifying which set was used to compose the document. Text, moreover, is the primary format used with such protocols as HTTP and SMTP, so XML needs no special handling when used with the most widely employed protocols of the Internet.

XML is also experiencing a wave of popularity at the moment. Technologies built around XML provide powerful capabilities useful to e-commerce applications. We will see one of those technologies – the styling and transformation language XSLT – later in the presentation. Because of the technical advantages and market popularity, XML tools are readily available on most platforms and for most programming languages. The result is that a critical mass of trading partners and suppliers is forming. XML is rapidly transitioning from a technical frontier to the safe choice for data exchange in e-commerce applications.

Getting Data into XML

Unless you are developing a brand new application that emits XML as its native export format, you will need to get your business data into XML. Some sources of data are:

- Data manually entered into an HTML form
- Relational data from a commercial database
- Data exported from an application in a flat file format such as CSV or positional data
- Data transmitted via messaging middleware in a flat file format

The flat file sources may be the result of data dynamically created by an application or exported from some storage form, e.g. a database or a spreadsheet, that is capable of translation from its native format to a common flat file format.

Regardless of the source, we need convenient, ready-built tools for creating XML. This is the starting point for connecting e-commerce applications to XML.

ADO Recordsets

Since version 2.1, ADO recordsets have been capable of exporting from the native binary form of a recordset to XML. ADO 2.1 permitted recordsets to be saved to a disk file in XML form. ADO 2.5 broadened this capability so that the XML data can be written to an `IStreamPersist` interface. This means that a recordset can be written as an XML document directly to an instance of the MSXML parser, the body of a Microsoft Message Queue (MSMQ) message, or the Response object in ASP. This opens many possibilities for writing e-commerce applications that depend on a relational database on the Windows platform.

The sample in the presentation shows the results of a query against the publishers table of the SQL Server pubs database. It consists of two main segments. The first is an embedded XML schema. This describes the structure of the XML document itself. A moment's thought will show you why this is absolutely necessary. The data requested by a SQL query can be different from every other SQL query. This results in different results that need to be expressed. There is no way one XML vocabulary can express all possible result sets.

The second major section of the XML document is the data itself. The format favored by ADO is a loose form, consisting, in the most commonly used case, of one element per row. The columns of the recordset are expressed as attributes of the element. Nested recordsets, involving a column that is itself a recordset, are expressed as elements nested within the row element. Here's a very brief excerpt from the sample:

```
<rs:data>
  <z:row pub_id="0736" pub_name="New Moon Books" city="Boston"
       state="MA" country="USA" />
  <z:row pub_id="1389" pub_name="Algodata Infosystems"
       city="Berkeley" state="CA" country="USA" />
</rs:data>
```

RDBMS Support

ADO support for XML was quickly followed by support for XML in the major RDBMS's themselves. Oracle 8 introduced support, quickly followed by a Microsoft technology preview for SQL Server 7.0, and SQL Server 2000. The basic format used is similar in most cases to that used by ADO. The element per row, attribute heavy form, sometimes known as the **canonical form**, is useful and easily implemented from a database cursor. Alternatives to this form supported by SQL Server 2000 include making the attributes child elements of an element named for the table, as well as hybrid forms in which the query explicitly designates which columns become elements and which remain attributes. Here is the sample given above rendered in the form in which columns are child elements:

```
<root>
  <publisher>
    <pub_id>0736</pub_id>
    <pub_name>New Moon Books</pub_name>
    <city>Boston</city>
    <state>MA</state>
    <country>USA</country>
  </publisher>
```

```
  <publisher>
    <pub_id>1389</pub_id>
    <pub_name>Algodata Infosystems</pub_name>
    <city>Berkeley</city>
    <state>CA</state>
    <country>USA</country>
  </publisher>
</root>
```

Transforming Data

When we speak of transforming data, we are not talking about changing our data into something completely different. Rather, we have the problem of two dissimilar but related XML vocabularies. Related, in the sense that the two vocabularies are talking about the same thing. If they weren't, there would be no need to transform them, nor would we be able to do so. Dissimilar, in the sense that the form of the data is a bit different. Consider the following:

```
<Contact fname="John" lname="Doe"/>
```

```
<Person role="contact">
  <Name>
    <First>John</First>
    <Last>Doe</Last>
  </Name>
  <id>12345-asd</id>
</Person>
```

Both XML fragments are clearly expressing the name of a person. In the first, it has been made clear from the element name that we are talking about a contact, although it is not clear what sort of contact we are talking about. Perhaps this is the person to call in the event that a shipment is delayed, or perhaps we are talking about a sales contact. In the second fragment, we are dealing with a person, and that person's status as a contact is expressed in the value of the `role` attribute.

To relate the second to the first, we need to assign the values of the `Name` related elements to the attributes of the `Contact` element in the first fragment, dropping the `id` element entirely. This can be done with an XML-related technology called the **Extensible Stylesheet Language – Transformation** (**XSLT**). XSLT is a data-driven method for converting one XML document into another. An XSLT stylesheet contains a series of rules, called templates, each of which specifies a pattern to be matched in the **source** document and a set of actions to be taken when the pattern is matched. The results of the actions become the body of the **target** document. You start with a source document, apply the stylesheet actions, and generate the target document. You can see how this might be useful. We have a business document in one XML vocabulary, and we want its equivalent in another vocabulary. I write a stylesheet describing the transformation we need to make, and we can create the needed document. XSLT only became a W3C Recommendation last year, but a number of XML parsers support it, so we should have no trouble finding one that will do the job for us. The preview release of Microsoft's parser supports XSLT, as do a number of parsers and editors from IBM's alphaWorks laboratory. Writing a stylesheet is generally easier than writing all the custom code to perform the transformation ourselves.

The Basic Transformation Process

The basic process of setting up and executing a transformation on B2B documents is as follows. First, you need the schemas or Document Type Definitions (DTDs) for the two documents. You control one document, so getting the information on its structure should be easy. You may need to contact the owner of the other document to get information on its structure, or you

may be able to get the information from a third party. A number of portal sites exist, such as BizTalk.org, with the hope of becoming a clearinghouse for just this sort of information.

Next, you need to determine the mapping between the two schemas. Strictly speaking, you don't absolutely need the schemas for the two vocabularies, but you do need to be sure that you know all the possible elements, attributes, and structures that can appear in documents of the two types. A comprehensive sample of a document written to each vocabulary might be sufficient, but generally speaking you will want to get a schema or other documentation from the vocabulary originator. These two steps – determining the vocabularies needed and developing a mapping – occur at design time.

When you have an actual source document instance that requires transformation, your code loads the document and a stylesheet into your XSL processor. Using the Windows XML parser, you load the source document into one parser and the stylesheet into another:

```
doc = new ActiveXObject("MSXML2.DOMDocument");
stylesheet = new ActiveXObject("MSXML2.DOMDocument");

doc.load("OrderRequestSample.xml");
stylesheet.load("cxmlToPO.xsl");
```

Next, you apply the stylesheet – XSLT stylesheets are themselves XML documents – to the source document by calling the `transformNode` method on a node in the source document:

```
StrXML = doc.transformNode(stylesheet);
```

The node you select is the starting point for the transformation. If you are transforming the entire document, you call the method on the entire source document. The result is a new XML tree describing the target document. With this in hand, you can save it to disk or transmit it to the recipient. Let's try a practical example of this process.

cXML to XML Common Purchase Order

The cXML vocabulary is a consortium (http://www.cxml.org) effort to describe common business documents in XML. Messages in this vocabulary consist of documents whose root element is named cxml, and whose body can contain a number of different elements. These elements are what distinguish the various messages possible in the cXML vocabulary.

For its part, Microsoft has created a number of XML vocabularies for use with its BizTalk Server product. These have the same aim as cXML, but unlike that effort, each message is a different XML document type.

One of these types is the Common Purchase Order (PO). cXML has a message type called `OrderRequest` which is analogous to a purchase order.

There are many vocabularies with similar goals in existence today. It is worth checking one of the repository sites, such as BizTalk.org or RosettaNet, to see if one of their vocabularies meets your needs. Your needs and data requirements are the determining factor.

To test our XSLT transformation process, we will attempt to devise a mapping between cXML messages containing an `OrderRequest` element and the Common PO vocabulary.

cXML OrderRequest

A cXML `OrderRequest` message is a purchase order. It is capable of expressing a new order as well as updates to existing ones. All cXML messages include a general `Header` element,

followed by a Request element that, in turn, contain the element that denotes the type of message we are sending. In our case, this is OrderRequest. The OrderRequest element contains a header and one or more ItemOut elements which correspond to the line items in a PO. To simplify matters, we will assume we are dealing with a new PO rather than a change to an existing one. Take a look at the sample OrderRequest message found in the file OrderRequestSample.xml. Here's a brief excerpt:

```
<cXML payloadID="3223232@ariba.acme.com"
    timestamp="2000-03-12T18:39:09-08:00">
  <Header>
      <From>
       . . . .
      </From>
      <To>
       . . . .
      </To>
        . . .
  </Header>
  <Request deploymentMode="test">
      <OrderRequest>
           <OrderRequestHeader orderID="DO1234"
                       orderDate="2000-03-12" type="new">
               <Total>
                   <Money currency="USD">50.00</Money>
               </Total>
               <ShipTo>
                . . . .
               </ShipTo>
               <BillTo>
                . . . .
               </BillTo>
               <Shipping trackingDomain="FedEx"
                         trackingId="1234567890">
                   <Money currency="USD">0.00</Money>
                   <Description>
                      <ShortName>FedEx 2-day</ShortName>
                   </Description>
               </Shipping>
               <Tax>
                . .
               </Tax>
               <Payment>
                . . . .
               </Payment>
               <Contact role="sales">
                  <Name>John Doe</Name>
               </Contact>
           </OrderRequestHeader>
           <ItemOut quantity="2"
                    requestedDeliveryDate="2000-03-12"
                    lineNumber="001">
               <ItemID>
                   <SupplierPartID>1233244</SupplierPartID>
               </ItemID>
               <ItemDetail>
                  <UnitPrice>
                      <Money currency="USD">25.00</Money>
                  </UnitPrice>
                  <Description>
```

```
                    <ShortName>AtomWidget</ShortName>
                  </Description>
                    . . .
                  <ManufacturerPartID>234</ManufacturerPartID>
                    . . .
                </ItemDetail>
                <ShipTo>
                 . . . .
                </ShipTo>
                  . . .
            </ItemOut>
        </OrderRequest>
  </Request>
</cXML>
```

XML Common PO

The Common PO is a bit simpler than the `OrderRequest` message. It is provided with the BizTalk Server product as an XML counterpart to EDI purchase orders. It contains a root element of `CommonPO`, followed by a number of elements like `POHeader`, `BillTo`, and `ShipTo`, followed by one or more `Item` elements and then a `Total` element. Each `Item` element is a line item in the purchase order.

Transformation

We'll first try the transformation using an XSLT stylesheet and just enough custom code to load the source document and stylesheet and execute the transformation. We already know one mapping: `ItemOut` in cXML contains the information we will need in `Item` elements of the Common PO. Another mapping you can determine from the stylesheet is `Shipping` to `CarrierDetail` (to pick up the shipping carrier). One interesting detail is that there are several calculated values. `QuantityTotal` and `LineItemTotal` do not correspond to any element or attribute in the `OrderRequest` document, but must be determined from looking at the source document and counting the line items and totaling the quantity attributes.

The Stylesheet

The stylesheet, `cxmltoPO.xsl`, contains all the mappings. Note that it is an XML document that is rooted by the `xsl:stylesheet` element. The templates alluded to earlier are contained in the `xsl:template` elements. Without going into a great lecture on the fundamentals of XSLT, we can say a few things to get you started. Each template contains a match attribute whose value is an XPath expression denoting some XML structure that is to be matched. XPath is a W3C Recommendation that describes how to query within an XML document, based on a path from a known XML node and some desired properties or conditions. If a match is found, everything within the template is executed to create output XML. There is a starting template, denoted by a match value of `/`, which is the XPath expression for the root of the document. The `xsl:apply-templates` element tells the XSLT processor to continue with the pattern in its select attribute. In our case, we are directing it to the cXML element. This is an important element as, without it, the processor would cease working after matching the root. Whenever we want to continue with another template, we need to insert an `xsl:apply-templates` element. In our stylesheet, you will also see some looping structures denoted by the `xsl:for-each` element. That is a template that is to be applied for every part of the XML that matches the XPath expression in the select attribute, i.e. iterate over each element that matches. Let's explain a fragment from the stylesheet:

```
<xsl:template match="cXML">
  <CommonPO>
```

```
    <xsl:apply-templates select="Request"/>
    <FOB ShipPaymentMethod="CC"/>
    <SpecialCharge Type="X" HandlingMethod="XX"/>
    <TermsOfSale PaymentMethod="PCard"/>
    <DateReference Description="XYZ"/>
    <xsl:apply-templates
     select="Request/OrderRequest/OrderRequestHeader/Shipping"/>
```

Upon matching the cXML element, this template writes a CommonPO element to begin our CommonPO output document. Next, it invokes the template that handles the Request element. After that template has generated its output, we generate hard-coded elements named FOB, SpecialCharge, TermsOfSale, and DateReference. This is because we have fixed their values for our purposes. Next, the template for reaching down deep into the document and handling the Shipping element is invoked.

Now, to handle the calculated values we need some script. XSLT has its own method of dealing with this that may be different from what you are used to. It uses parameterized templates. Our stylesheet uses two proprietary elements supported by the Microsoft parser since version 2.0, xsl:eval and xsl:script. These permit us to include a CDATA section with JavaScript code that uses the document object model to do our calculations.

The Code

Given the stylesheet from design time, how do we perform a runtime XSLT transformation? Here is the entire code you would use in a client-side script minus some error handling. We begin by setting up two instances of the Microsoft parser, preview edition:

```
doc = new ActiveXObject("MSXML2.DOMDocument");
stylesheet = new ActiveXObject("MSXML2.DOMDocument");
doc.async = false;
stylesheet.async = false;
```

Since this is the preview version, I've taken care to provide the version specific progid. After I have my two instances, I tell each to perform synchronously as I am writing a scripted HTML page and cannot easily do multithreading. Now I load my source document and the XSLT stylesheet:

```
doc.load("OrderRequestSample.xml"); stylesheet.load("cxmlToPO.xsl");
```

Now I want to call the transformNode method on the document itself (not the documentElement property). The result of this call is XML text, so I'll pass that directly to the alert method in JavaScript to pop the result up in a dialog box for inspection. If I needed to manipulate the target document further, I could call the Microsoft-proprietary transformNodeToObject method and get a DOM tree in return.

```
alert(doc.transformNode(stylesheet));
```

Is There a Better Way?

Our simple case was harmless enough, although it can't handle flat file formats. More complicated transformations could prove difficult, however. We need to be fluent in XSLT, and writing and debugging stylesheets can prove time-consuming. While it is better than writing custom code for transformations, we can't help but wonder if there's a better way. If we are going to have lots of partners requiring lots of transformations, we'd rather not have to write a lot of stylesheets.

What we need is a tool that will accept a message and do a transformation for us. Rather than write the stylesheet, we'd like the tool to generate the stylesheet based on some input from us. Ideally, we'd like something with extensions that could handle flat file formats in addition to XML. Happily, there are a number of such products.

B2B Messaging Products

Some B2B products of interest in our situation primarily work to the data transformation aspect of B2B processing. This will be most useful to you when you have large quantities of legacy data to be transformed to disk or when you have an unusual business process requiring custom programming.

More advanced products also include some form of workflow processing. Data transformation is one possible action that can be taken at any step in the workflow. Other actions include decision points and message reception or transmission. Such products give you the capability to map out complex business processing with little or no programming on your part.

Microsoft BizTalk Server

Once such product, which will be demonstrated shortly, is Microsoft BizTalk Server. This is a completely new product due imminently. It consists of graphical design time tools for specifying B2B vocabularies and data transformation mappings, as well as runtime servers for message transformation and workflow scheduling. While you can extend the product through code, it is designed to be data driven. This means that you will primarily work with graphical interfaces to specify configurations and mappings to be applied by the server at run time. As messages pass through the system, you have the option of creating an audit trail with information taken from each message.

BizTalk Mapper

One of the tools is the BizTalk Mapper. It takes two message specifications (BizTalk's generic construct for XML schemas and flat file message descriptions) and lets you point and click links between related items. Mapper uses the terms record and field. Not only is this more familiar to the average business programmer, but it avoids forcing flat file programmers to think in terms of XML. In addition to direct links, we can drop intermediate elements, called **functoids**, onto the mapper grid. We specify a link coming from the source document into the functoid, and a link coming out to the target. Functoids are used to perform some intermediate processing that is not supported by the core elements of XSLT, such as numeric type conversions, string manipulation, and database access.

cXML to XML Common PO via BizTalk

Returning to our example, we import the cXML DTD into BizTalk's Editor to create a message specification for that message type. BizTalk includes a specification for the Common PO. Next, we load these specifications into the Mapper and create the links you saw in the demonstration. Note the use of functoids to obtain the calculated values. Although there is a script function that offers you the ability to enter your own scripts in JavaScript or VBScript, we did not need to resort to that. There are preexisting functoids for the sums we need, so it is simply a matter of adding them to the grid. The code to perform the iterations and calculate the sums is generated by the functoid.

The Code

Here is every line of code used in our sample to submit the cXML sample for transformation:

```
Dim IntC As New Interchange
. . .
```

```
sDocument = CustMessage.Text

sHandle = IntC.Submit(MODELDB_OPENNESS_TYPE_NOTOPEN, sDocument, ,
                      "Organization Name", "Home Organization",
                      "Organization Name", "XPurchasing")
```

The first line retrieves the text of the message from the edit control in our application. In a real application you would replace this line with code to obtain the message from some incoming or outbound message protocol. The object `IntC` is an `Interchange` object. This is an interface that is part of BizTalk Server and it acts as your gateway into the server's message queue. We are asynchronously submitting the message with information telling BizTalk who is to receive it following transformation. In this case, we, the Home Organization, are sending it to the XPurchasing organization. Based on this information, BizTalk looks at the message type and looks for a configuration that matches. Once found, it performs the transformation specified.

Configuration

To make this example work, we created the XPurchasing organization and specified some information through the BizTalk Management Desk. We told BizTalk that XPurchasing uses the Common PO message type. Using the BizTalk Management Desk tool, we created a port, which is the combination of a receive location and a message type (think "port for passing the message to the recipient"). For the example, we said we wanted to save POs to a disk location. Next, we created a Channel, which is an agreement for a message type and a port. Channels specify the format of the message coming into the port, the desired output format, and any specialized routing or security provisions for messages transmitted through the port, under the guidelines of the channel. We said that this port should receive cXML `OrderRequest` messages, transform them into Common PO messages, and save the result to disk as specified by the port. All this took about five minutes using the Management Desk. More complicated cases would take slightly longer, but you could provide for message encryption and authentication, auditing, and specify other protocols like HTTP or MSMQ.

Flat File Processing

We mentioned that XSLT won't handle flat file source documents and may not be able to produce flat file output. BizTalk uses a series of components to allow the XSLT engine to work with such formats. This is also how BizTalk supports EDI message formats. The schematic depicted in the slides shows the case in which you want to transform a flat file source document into a flat file target document. A flat file parser uses the message specification for the source document to locate the records and fields and transform the document into an internal XML representation. Next, an XSLT stylesheet transforms that document into another XML representation that more closely resembles the target flat file format. Finally, a flat file parser component uses the target message specification to transform the intermediate XML format into the target flat file format. This process harnesses the XSLT engine but lets you use flat files or mix them with XML documents in any combination.

Orchestration

A simple message exchange such as we described requires nothing further. Your real-world B2B processes, by contrast, will usually require a series of messages to complete. You might send out a request for bids, receive responses, select a winner, and transmit the order. Your supplier must acknowledge the order, provide status updates, and send a message indicating that the product has shipped. The Orchestration feature of BizTalk lets you set up a complex workflow like this graphically, then uses that configuration to drive the server.

A BizTalk orchestration document is a series of actions and decision points. Message exchanges are modeled as ports. A port is connected to some component or message. This connection maps to a channel in BizTalk. Within the orchestrated workflow, you can have concurrency, or

you may suspend processing to await an incoming message. The graphical tool generates an XML document that is loaded by the server. This describes a state machine that the server is to execute. The mappings, ports, and channels you define are coordinated by the state machine, at the points specified in the workflow, to accomplish the overall business process. Applications and services can invoke the workflow by referring to the file generated by the Orchestration tool in a COM moniker.

B2B Portals

Performing B2B messaging with a product like BizTalk Server requires information from your trading partners. Minimally, you need the message specifications for the formats they use, as well as information on the messaging protocols and configuration needed to talk to them. You will also want to establish security arrangements as well as any non-technical contractual items required by your respective organizations. For a limited number of partners, you can get this information directly. For larger numbers, or for ad hoc exchanges, you need a portal.

The vision of a messaging portal is to help companies locate appropriate partners and give them the information they need to begin communicating. In return, these portals might take a transaction fee. BizTalk.org is Microsoft's free portal containing a searchable repository of message schemas. RosettaNet is a consortium that is developing a number of standard workflows. It too, offers a repository of messaging information. Trading hubs and portal sites are being created for a number of vertical markets using these and competing e-commerce frameworks and technologies.

Actual practice is diverging from the vision, at least initially. A number of vertical portals have been created or are being developed to help groups of mammoth manufacturers exchange messages with their suppliers. Examples of these may be found in the automotive and aerospace industries. Such manufacturers can get away with controlling the portal because their size creates immediate critical mass and because the size of their business allows them to dictate standards to their suppliers. It remains to be seen if this will spread to the rest of the business world or whether third-party portals will have a chance.

Summary

If you are building a B2B site on the Web, design it with automated message interchanges in mind. Use XML as the exchange medium. Think of the business process as a sequence of applications connected by XML messages. Even if you are using XML, message transformation will be a necessity for you.

Use existing tools to get your platform-specific data into XML and to accomplish data transformation. If you anticipate large numbers of partners and transformations, look into third party messaging and transformation tools. The degree of sophistication you require will be dictated by the needs of your site. In advanced cases, look for tools capable of implementing business process workflows as an alternative to custom code. They will be much more productive for you, and you will be better able to respond to changes in the business process.

Further Resources

cXML

http://www.cXML.org

BizTalk

Framework, repository: http://www.biztalk.org/BizTalk/default.asp
Server: http://www.microsoft.com/biztalkserver

RosettaNet

http://www.rosettanet.org

XSLT

Recommendation: http://www.w3.org/TR/xslt
Tools and programming: http://msdn.microsoft.com/xml/default.asp