Merrill Communications LLC
d/b/a Merrill Corporation
Exhibit 1006 pt. 2

Chapter 6

An online auction is the epitome of a complex real-time interactive application, so the Microsoft Web Applications Team built a realistic *Auction Demo* to show how simply one can be implemented as an XML three-tier Web application. This chapter is sponsored by Microsoft Corporation, `http://www.microsoft.com`, and was prepared by Charles Heinemann.

The *Auction Demo* is a three-tier Web application that simulates an online auction using technologies that are available in *Internet Explorer 4.0* (IE 4.0). It allows you to view the items available for auction, place bids on those items, and monitor the bids placed by fellow bidders.

Like other three-tier Web applications, the Auction Demo has data sources on the back end, a user interface on the client, and a Web server in the middle. We'll see how it was developed, using just three permanent Web pages:

userInterface.htm

This page uses *Dynamic HTML* (DHTML) to allow the Web browser to present the auction information to the user. It contains scripts that collect or update data on the middle tier by requesting *Active Server Pages* (ASP).

auction.asp

This page is an ASP file. When userInterface.htm requests this page, the scripts in it are executed on the server. The scripts

generate auction.xml, an XML document that contains the latest auction data, which is delivered to the client.

makebid.asp

This page is requested by userInterface.htm when the user wants to make a bid. It is executed on the middle tier, causing the data source to be updated with the new bid information.

The user interface (UI) for the *Auction Demo* is shown in Figure 6-1. It is the rendition of the userInterface.htm *Dynamic HTML* page, which is downloaded to the client when the user clicks on a link to the auction.

That page has scripts within it that handle all the client-side activity. That includes requesting data from the middle tier in order to display the most current values of the items and bids. We'll see later how the UI page does its thing, but first let's look at how the middle tier collects and transmits the data. It does so by packaging the data as XML documents.

6.1 | Getting data from the middle tier

The role of the middle tier in a Web application is to gather information from data sources and deliver it in a consistent manner to clients. In the Auction Demo we start with a single data source, an ODBC-compliant database. (Later we'll see how multiple data sources of different kinds can be accessed.)

The "Auction" database used for the *Auction Demo* is a relational database with two tables, an "Item" table and a "Bids" table. The "Item" table contains data about each of the items up for auction. It is shown in Figure 6-2.

For the sake of clarity, we'll just cover the "Item" table in this chapter (the "Bids" are handled similarly). You can see the full demo at `http://www.microsoft.com/xml`. We want to deliver the data in that table in the form of an XML document, so the client's user interface page won't have to know anything about the actual data source.

©1998 THE XML HANDBOOK™

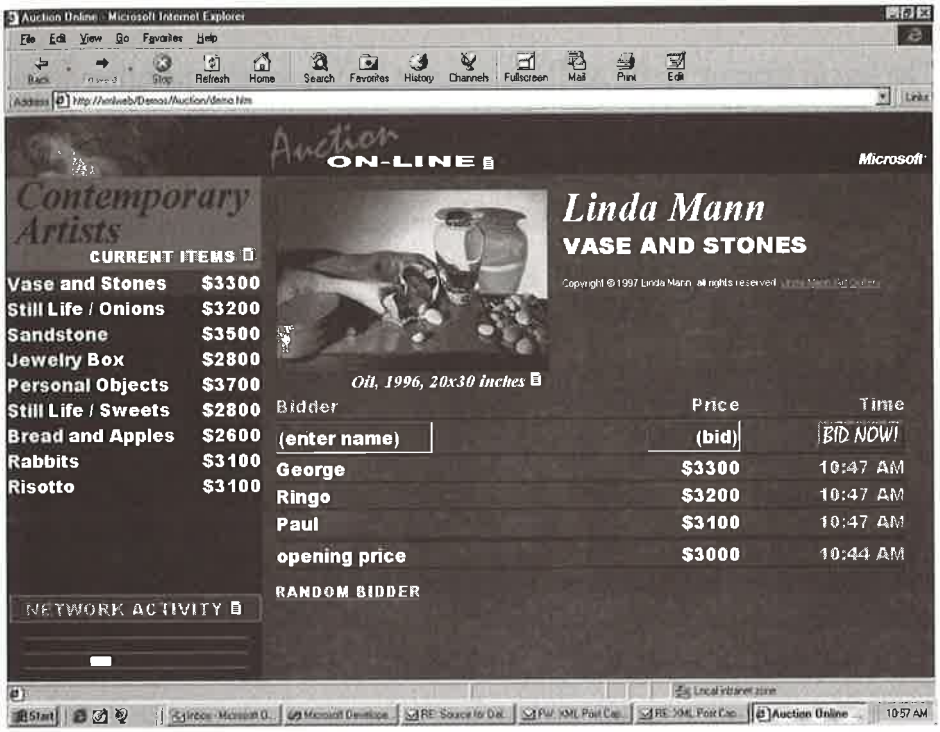


Figure 6-1 The *Auction Demo* user interface.

Title	Artist	Dimensions	Materials	Year
Vase and Stones	Linda Mann	20x30 inches	Oil	1996
Still Life / Onions	Linda Mann	20x30 inches	Oil	1997
Sandstone	Linda Mann	20x30 inches	Oil	1995
Jewelry Box	Linda Mann	20x30 inches	Oil	1994
Personal Objects	Linda Mann	20x30 inches	Oil	1995
Still Life / Sweets	Linda Mann	20x30 inches	Oil	1994
Bread and Apples	Linda Mann	20x30 inches	Oil	1995
Rabbits	Linda Mann	20x30 inches	Oil	1996
Risotto	Linda Mann	20x30 inches	Oil	1995

Figure 6-2 The *Auction Demo* item table.

6.1.1 *Defining the XML document structure*

The key to creating useful XML documents is the proper structuring of the data. For the *Auction Demo*, that means deciding how a record in the "Item" table will be represented as an ITEM element in XML. There is a straightforward mapping, shown in the following data-less element:

Example 6-1. Template for an ITEM element.

```
<ITEM>
  <TITLE></TITLE>
  <ARTIST></ARTIST>
  <DIMENSIONS></DIMENSIONS>
  <MATERIALS></MATERIALS>
  <YEAR></YEAR>
</ITEM>
```

For each field in the "Item" table, there is a corresponding subelement of the ITEM element.

To generate XML documents with these ITEM elements, the *Auction Demo* uses ASP files.

6.1.2 *Using ASP files to generate XML documents*

XML can be generated on the middle tier using *Active Server Pages*. ASP offers an environment in which Web authors can create documents dynamically by intermixing markup languages with in-line scripts. The scripts can be written in a variety of scripting languages, including *JScript* and *VBScript*, and can invoke server-side components to access databases, execute applications, and process information.

When a browser requests an ASP file, it is first processed by the server, which delivers a generated Web page containing standard markup.

In an ASP file, commands and scripts are delimited by "<%" and "%>". Everything not so delimited is markup or data that will appear in the generated page. For example, consider the following trivial ASP file:

The file, after establishing that the scripting language is *VBScript*, creates the variable "Total" with the value "2". The following line generates an

Example 6-2. Sample ASP file.

```
<%@ LANGUAGE = VBScript%>
<%DIM Total = 2%>
<AMOUNT><%=Total%></AMOUNT>
```

XML "AMOUNT" element whose content is generated by executing the small script, which in this case retrieves the value of "Total".

When the browser requests this file, it will actually receive the XML document that is generated from the file, as shown in Example 6-3:

Example 6-3. XML document generated by sample ASP file.

```
<AMOUNT>2</AMOUNT>
```

Note that the ASP syntax (<% . . . %>) does not cause an XML parsing error. That is because the ASP file is not itself an XML document. The ASP file is processed on the server and only the generated XML document is returned to the client.

In the case of the *Auction Demo*, the file auction.asp is used to access the "Auction" database and generate XML containing the data within the "Item" and "Bids" tables. The ability to generate XML on the middle tier makes it possible to provide the Web application with content that can be manipulated on the client and refreshed without having to refresh the entire user interface.

In Example 6-4, auction.asp begins like the ASP file in Example 6-2, by declaring the scripting language. The next two lines are the XML declaration and the start-tag of the root element (AUCTIONBLOCK) of the XML document to be generated, which we will call "auction.xml".

Example 6-4. Start of auction.asp.

```
<%@ LANGUAGE = VBScript %>
<?XML VERSION="1.0"?>
<AUCTIONBLOCK>
```

Next, a connection to the "Auction" database is established and that connection is opened:

Example 6-5. Connecting to the database.

```
<%
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open "Auction","Auction","Auction"
%>
```

A "record set" variable (ItemRS) is now established to contain each record of the "Item" table as it is accessed, and a "Do While" loop is begun to perform the access.

Example 6-6. Preparing to access the "Item" records.

```
<%
Set ItemRS = Conn.Execute("select * from item")
Do While Not ItemRS.EOF
%>
```

Next, the template in Example 6-1 is used to create the XML ITEM element that will be generated. Just as in Example 6-2, a small script is inserted as the content of each subelement of ITEM within auction.asp. In this case, the script extracts the corresponding field's data from the record set.

Example 6-7. Markup and scripts to generate an ITEM element.

```
<ITEM>
  <TITLE><%=ItemRS("Title")%></TITLE>
  <ARTIST><%=ItemRS("Artist")%></ARTIST>
  <DIMENSIONS><%=ItemRS("Dimensions")%></DIMENSIONS>
  <MATERIALS><%=ItemRS("Materials")%></MATERIALS>
  <YEAR><%=ItemRS("Year")%></YEAR>
</ITEM>
```

After an ITEM element is generated, the script moves to the next record in the record set. The loop is then repeated. Once all of the records have been run through, the root element is ended.

The complete auction.asp file is in Example 6-9.

Example 6-10 is an abridged version of the XML document (auction.xml) generated by the auction.asp file in Example 6-9.

Example 6-8. Repeating the loop and ending the document.

```
  <%
  ItemRS.MoveNext
  Loop
  %>
</AUCTIONBLOCK>
```

Example 6-9. The complete auction.asp file.

```
<%@ LANGUAGE = VBScript %>
<?XML VERSION="1.0"?>
<AUCTIONBLOCK>
<%
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open "Auction","Auction","Auction"
Set ItemRS = Conn.Execute("select * from item")
Do While Not ItemRS.EOF
%>
  <ITEM>
    <TITLE><%=ItemRS("Title")%></TITLE>
    <ARTIST><%=ItemRS("Artist")%></ARTIST>
    <DIMENSIONS><%=ItemRS("Dimensions")%></DIMENSIONS>
    <MATERIALS><%=ItemRS("Materials")%></MATERIALS>
    <YEAR><%=ItemRS("Year")%></YEAR>
  </ITEM>
  <%
  ItemRS.MoveNext
  Loop
  %>
</AUCTIONBLOCK>
```

Example 6-10. Abridged auction.xml document generated by auction.asp.

```
<?XML VERSION="1.0"?>
<AUCTIONBLOCK>
  <ITEM>
    <TITLE>Vase and Stones</TITLE>
    <ARTIST>Linda Mann</ARTIST>
    <DIMENSIONS>20 X 30 inches<DIMENSIONS>
    <MATERIALS>Oil</MATERIALS>
    <YEAR>1996</YEAR>
  </ITEM>
  <ITEM>
  . . .
  </ITEM>
  . . .
 </AUCTIONBLOCK>
```

6.1.3 *Generating XML from multiple databases*

One powerful reason to generate XML documents on the middle tier is that they can contain data that is sourced from multiple independent databases. The technique is similar to what we've already seen. The only difference is that multiple database connections are made instead of one.

The ASP file in Example 6-11 does just this, generating a single XML document with data from the databases "Gallery1" and "Gallery2".

The XML generated by the ASP file in Example 6-11 looks structurally just like Example 6-10, an AUCTIONBLOCK element with multiple ITEM children. However, the data content originates from two different data sources.

Also notice that, for the DIMENSIONS, MATERIALS, and YEAR elements, the source fields in the "Gallery2" database are actually labeled differently from the corresponding fields in "Gallery1." One benefit of consolidating the data on the middle tier is that the semantics can be identified consistently, and therefore made more easily accessible.

6.1.4 *Generating XML from both databases and XML data sources*

The middle tier can source data of different kinds, not just databases. In Example 6-11, the ASP file, as in previous examples, first accesses data from "Gallery 1", an ODBC compliant database. However, it then adds data from "Gallery 3", a source of XML documents.

The Gallery3 XML document is processed by the MSXML parser (details below), which allows access to the document's data content. Note that there is no way – and no need – to tell whether Gallery3 is a persistent document, or was generated by another middle-tier application.

Also, look at the YEAR element. Just as with the Gallery2 database in the previous example, the original semantic label – in this case the DATE generic identifier – is changed on the middle tier to ensure consistency.

©1998 THE XML HANDBOOK™

Example 6-11. Generating one XML document from two databases.

```
<%@ LANGUAGE = VBScript %>
<?XML VERSION="1.0"?>
<AUCTIONBLOCK>
<%
'The connection to the Gallery1 data source is made
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open "Gallery1","Gallery1","Gallery1"
Set ItemRS = Conn.Execute("select * from item")
Do While Not ItemRS.EOF
%>
  <ITEM>
    <TITLE><%=ItemRS("Title")%></TITLE>
    <ARTIST><%=ItemRS("Artist")%></ARTIST>
    <DIMENSIONS><%=ItemRS("Dimensions")%></DIMENSIONS>
    <MATERIALS><%=ItemRS("Materials")%></MATERIALS>
    <YEAR><%=ItemRS("Year")%></YEAR>
  </ITEM>
  <%
  ItemRS.MoveNext
  Loop
  %>
<%
'The connection to the Gallery2 data source is made
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open "Gallery2","Gallery2","Gallery2"
Set ItemRS = Conn.Execute("select * from item")
Do While Not ItemRS.EOF
%>
  <ITEM>
    <TITLE><%=ItemRS("Title")%></TITLE>
    <ARTIST><%=ItemRS("Artist")%></ARTIST>
    <DIMENSIONS><%=ItemRS("Size")%></DIMENSIONS>
    <MATERIALS><%=ItemRS("Medium")%></MATERIALS>
    <YEAR><%=ItemRS("Date")%></YEAR>
  </ITEM>
  <%
  ItemRS.MoveNext
  Loop
  %>
</AUCTIONBLOCK>
```

©1998 THE XML HANDBOOK™

Example 6-12. Generating one XML document from a database and another XML document.

```
<%@ LANGUAGE = VBScript %>
<?XML VERSION="1.0"?>
<AUCTIONBLOCK>
<%
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open "Gallery1","Gallery1","Gallery1"
Set ItemRS = Conn.Execute("select * from item")
Do While Not ItemRS.EOF
%>
  <ITEM>
    <TITLE><%=ItemRS("Title")%></TITLE>
    <ARTIST><%=ItemRS("Artist")%></ARTIST>
    <DIMENSIONS><%=ItemRS("Dimensions")%></DIMENSIONS>
    <MATERIALS><%=ItemRS("Materials")%></MATERIALS>
    <YEAR><%=ItemRS("Year")%></YEAR>
  </ITEM>
  <%
  ItemRS.MoveNext
  Loop
  %>
<%
'Here the connection to the Gallery3 data is made
Set XML = Server.CreateObject("msxml")
XML.URL = "http://datasource3/Gallery3.xml"
Set Items = XML.root.children
For I = 0 to Items.length - 1
%>
  <ITEM>
    <TITLE><%=Items.item(I).children.item("TITLE").text%>
    </TITLE>
    <ARTIST><%=Items.item(I).children.item("ARTIST").text%>
    </ARTIST>
    <DIMENSIONS><%=Items.item(I).children.item("DIMENSIONS").text%>
    </DIMENSIONS>
    <MATERIALS><%=Items.item(I).children.item("MATERIALS").text%>
    </MATERIALS>
    <YEAR><%=Items.item(I).children.item("DATE").text%>
    </YEAR>
  </ITEM>
<%Next%>
</AUCTIONBLOCK>
```

©1998 THE XML HANDBOOK™

6.2 | Building the user interface

The user interface is critical to the success of any application. It must allow the user to interact with the application in an efficient and straightforward manner. The user interface for the *Auction Demo* was built using DHTML.

DHTML is a set of features in *Internet Explorer 4.0* for creating interactive and visually interesting Web pages. It is based on existing HTML standards and is designed to work well with applications, *ActiveX* controls, and other embedded objects.

With DHTML a developer can create a robust and efficient UI without additional support from applications or embedded controls, or even return trips to the server. A *Dynamic HTML* page is self-contained, using styles and scripts to process user input and directly manipulate the HTML markup and other text within the page.

Let's see how userInterface.htm creates the Auction Demo interface by using scripts and the *IE 4.0 Document Object Model*. Two basic techniques are employed: procedural scripts and descriptive binding.

6.2.1 *Using procedural scripts*

Internet Explorer 4.0 includes the *MSXML* parser, which exposes the parsed XML document as a *document object model*.[1] Once exposed, scripts can access the data content of the XML elements and dynamically insert the data into the user interface.

The userInterface.htm code in Example 6-13 applies *MSXML* to auction.xml, the XML document generated by auction.asp. That creates an *ActiveX* object representing the parsed document.

Example 6-13. Creating the auction document object.

```
var auction = new ActiveXObject("msxml");
auction.URL = "http://Webserver/auction.asp";
```

1. The W3C is currently developing a common document object model for XML and HTML. There is a working draft on the CD-ROM. The *IE 4.0 Document Object Model* attempts to maintain compliance with the W3C draft as it evolves.

©1998 THE XML HANDBOOK™

In Example 6-14, the script next retrieves the root element. It then navigates the tree until it locates the TITLE element within the first ITEM element of auction.xml. The innerText property is used to insert the data content of TITLE into the user interface as the value of the "item_title" attribute, which appears on a DIV element.

Example 6-14. Extracting data from the auction document object.

```
var root = auction.root;
var item0 = root.children.item("ITEM",0)
var title = item0.children.item("TITLE").text;
document.all("item_title").innerText = title;
<DIV ID="item_title"></DIV>
```

One of the benefits of using procedural scripts to display XML documents is that you can manipulate the data content of an XML element before you display it. For example, if you wanted to display the dimensions of each painting using the metric system, rather than feet and inches, your script could simply convert the content of the DIMENSIONS element from inches to centimeters.

6.2.2 *Using descriptive data binding*

The *IE 4.0 XML Data Source Object* (XML DSO) is a declarative alternative to the procedural scripts described in the last section. The XML DSO is an applet (see Example 6-15) that enables the data of XML elements to be bound as the content of HTML elements.

Example 6-15. The *IE 4.0 XML Data Source Object* applet.

```
<APPLET ID=auction CODE=com.ms.xml.dso.XMLDSO.class MAYSCRIPT
      WIDTH=0 HEIGHT=0>
   <PARAM NAME="url" VALUE="auction.asp">
</APPLET>
```

In Example 6-15, the "url" parameter points the XML DSO to auction.asp, which causes auction.xml to be generated on the middle tier. A persistent XML source could also have been used.

In Example 6-16, data binding is used to populate the part of the user interface that shows the painting and the caption beneath it.

Example 6-16. Data binding with the XML DSO.

```
<TD>
  <DIV STYLE="margin-left:16px;
margin-top:16px;margin-right:16px">
    <DIV ID=pict></DIV>
      <DIV CLASS="details">
        <SPAN DATASRC=#auction DATAFLD=MATERIALS></SPAN>,
        <SPAN DATASRC=#auction DATAFLD=YEAR></SPAN>,
        <SPAN DATASRC=#auction DATAFLD=DIMENSIONS></SPAN>
      </DIV>
    </DIV>
  </DIV>
</TD>
```

With the XML DSO applet embedded in the Web page, no scripting is required to bind the data content of XML elements to HTML elements. Instead, the name of the document object (ID of the APPLET in Example 6-15) is specified as the value of the DATASRC attribute, and the generic identifier of the XML element is specified for the DATAFLD attribute.

One advantage of displaying XML with the XML DSO is that the XML document is processed asynchronously to the rendering of the page. Therefore, if the inventory of paintings were very large, the initial elements of the XML document could be displayed even before the last elements were processed.

6.3 | Updating the data source from the client

We have seen how userInterface.htm on the client obtained data to display to the user by invoking auction.asp on the middle tier. It can also enable the user to make his own bid by invoking another middle tier page, makebid.asp.

In the *Auction Demo*, the user bids by overwriting the price and bidder name in the first row of the bid table. A bid therefore consists of the "title"

©1998 THE XML HANDBOOK™

of the item currently displayed, the "price" of the new bid, and the name of the new "bidder".

These data items must be passed as parameters to makebid.asp, which executes a script to process them and update the database. The script returns to the client a "return message" XML document: a single element containing information about the status of the processing.

The script in userInterface.htm (see Example 6-17) begins by assigning the title of the current item up for auction to the "title" variable, the value of the "price" text box to the "price" variable, and the value of the "bidder" text box to the "bidder" variable.

It then creates the return message document object, which will state whether makebid.asp successfully updated the database. The three variables are passed as parameters to the ASP file when it is invoked.

Example 6-17. Sending a new bid to makebid.asp.

```
var title = current_item.children.item("TITLE").text;
var price = price.value;
var bidder = bidder.value;
var returnMsg = new ActiveXObject("msxml");
returnMsg.URL = "http://auction/makebid.asp?title=" +
  title + "&price=" + price + "&bidder=" + bidder;
```

In Example 6-18, makebid.asp (called by userInterface.htm in Example 6-17) assigns the values of the parameters "title", "price", and "bidder" to variables with the same names.

The "BidRS" record set object is then created and a connection to the "Auction" database is made. Note that the connection is made for both reading and writing. The "Bids" table is then opened and the new information is added to the record set, after which the connection is closed. The process is much the same as it was for auction.asp, except that the database is written to instead of just being read.

Finally, makebid.asp generates the return message document with the status of the update.

©1998 THE XML HANDBOOK™

Example 6-18. The makebid.asp file updates the database.

```
<%@ LANGUAGE = VBScript %>
<%
  title = Request.QueryString("title")
  price = Request.QueryString("price")
  bidder = Request.QueryString("bidder")

  Set BidRS = Server.CreateObject("ADODB.RecordSet")
  connect = "data source=Auction;user id=sa;password=;"
  BidRS.CursorType = 2
  BidRS.LockType = 3   ' read/write
  BidRS.Open "Bids", connect

  BidRS.AddNew
  BidRS("item") = title
  BidRS("price") = price
  BidRS("bidder") = bidder
  BidRS.Update
  BidRS.Close
%>
<STATUS>OK</STATUS>
```

6.4 | Conclusion

The entire *Auction Demo* was built using the methods described above. You can get a head start on building a similar Web application by modifying these scripts to suit your particular requirements.

XML enables Web applications by providing dynamic, accessible content that can be navigated and manipulated on the client. In addition, it enables the updating of content without having to refresh the entire user interface. This ability saves time by reducing round trips to the server for information that already exists on the client.

With XML, users can manage data over the Internet just as they presently do on their local machines. As a result, the Web is made a more interactive and interoperable medium. As the information superhighway is transformed into the data superhighway, Web applications similar to the

Auction Demo will allow for better utilization of the vast resources made available by the Web.

Analysis *The Auction Demo clearly illustrates the architecture of a three-tier application. It uses the middle tier as a transient data aggregator and normalizer. In other chapters you'll see different approaches to the middle tier, including persistent storage of metadata and the use of object paradigms rather than data paradigms.*

©1998 THE XML HANDBOOK™

XML and EDI: The new Web commerce

- Traditional EDI: Built on outdated principles
- Ubiquitous EDI: A quantum leap forward
- The New EDI: Leveraging XML and the Internet

©1998 THE XML HANDBOOK™

Chapter

7

XML and the Internet will dramatically reshape the Electronic Data Interchange (EDI) landscape. By driving down costs and complexity, EDI will become a truly ubiquitous technology that will reshape business as we know it. This introduction to EDI is sponsored by POET Software Corporation, `http://www.poet.com`, developers of object-oriented software for XML-based information systems. It was prepared by Mike Hogan.

Over the past several decades, corporations have invested trillions of dollars in automating their internal processes. While this investment has yielded significant improvements in efficiency, that efficiency has not been extended to external processes.

In effect, companies have created islands of automation that are isolated from their vendors and customers – their trading partners. The interaction among companies and their trading partners remains slow and inefficient because it is still based on manual processes.

7.1 | What is EDI?

Electronic Data Interchange (EDI) has been heralded as the solution to this problem. *EDI* is defined as the exchange of data between heterogeneous systems to support transactions.

EDI is not simply the exportation of data from one system to another, but actual interaction between systems. For example, Company B is a supplier to Company A. Instead of sending purchase orders, bills and checks in

hard copy form, the two might connect their systems to exchange this same data electronically.

In the process they could benefit in many other ways, including faster turnaround on orders, better inventory control, reduced financial float, complete real-time information about orders and inventory for improved decision-making, reduced costs for manual data input, and more. Companies that have implemented EDI rave about the various benefits.

In fact, these benefits can be expanded to a chain of suppliers. For example, Company C might be a supplier to Company B above. If companies B and C implement EDI, then Company A gains the additional benefits of superior integration with their entire *supply chain* of suppliers.

7.1.1 *Extranets can't hack it*

There is a significant gap between the business benefits described above and the actual implementation of EDI. This is because the actual implementation of "traditional EDI" is fundamentally flawed. It is difficult and costly to implement and, even worse, it requires a unique solution for each pair of trading partners. This situation is analogous to requiring a unique telephone line to be wired to each person to whom you wish to speak.

Many people falsely proclaimed the Internet as the solution to this problem. By implementing EDI over a single network, our problems would be solved. This "solution" was so exciting it was even given its own name, the extranet. Unfortunately, a network with a common protocol is still only a partial solution.

This is because the systems implemented in each company are based on different platforms, applications, data formats (notations), protocols, schemas, business rules, and more. Simply "connecting" these systems over the Internet does not, by itself, solve the problem. To use the phone system analogy again, this is analogous to wiring each business into the global phone network, only to realize that each company's phone system is unique, and incompatible with every other phone system.

And given the trillions of dollars companies have invested in automation, they are not simply going to replace these systems with new "compatible" solutions, assuming such things existed.

7.1.2 *XML can!*

The eXtensible Markup Language (XML) provides a solution for EDI over the Internet. XML is a universal notation (data format) that allows computers to store and transfer data that can be understood by any other computer system. XML maintains the content and structure, but separates the business rules from the data. As a result, each trading partner can apply its own business rules. This flexibility is critical to creating a complete solution for EDI.

There are additional technologies which are also part of the complete solution. Security, for example, is critical to EDI. Transactional integrity, connection stability, authentication and other services are also critical to implementing a complete solution. These requirements are addressed by technologies that are layered on top of the Internet. We refer to them generically as *Internet-based services.*

The final piece of the EDI solution is data storage. XML introduces a unique set of requirements for hierarchical naming and structure. It also requires rich relationships and complex linking. XML's use in EDI adds further requirements for metadata and versioning. These requirements levy heavy demands on database technology.

7.1.3 *The new EDI*

By combining XML, the Internet, Internet-based services and database connectivity, we have a complete solution for *New EDI.* Together, these technologies will not only change EDI, they will change our entire business landscape. EDI will metamorphose from a handful of unique interconnections, defined by the supply chain, into a "supply web". The supply web is an intelligent common fabric of commerce over the Internet.

According to Metcalfe's Law[1], the value of a network is roughly proportional to the number of users squared. Imagine what this means when your EDI "network" expands from a one-to-one proposition, to a true network that encompasses practically every company in the world. Suddenly, the trillions of dollars companies have invested in internal automation increase in value by several factors. By the same token, this information can also be

1. Robert Metcalfe is the creator of Ethernet.

©1998 THE XML HANDBOOK™

extended to customers, adding significant value to the vendor-customer relationship, thereby enhancing customer loyalty.

This is a pivotal time in the history of technology. With the emergence of XML, all of the pieces are available to create a universal mechanism for EDI. The Internet provides the transport. XML provides the flexible, extensible, structured message format. Various Internet-based services provide solutions for security, transactional integrity, authentication, connection stability, network fail-over and more.

Add to this sophisticated data storage and you have all of the pieces necessary to unite corporate islands of automation into a single coherent fabric of electronic commerce. This will result in dramatic improvements in efficiency, cost-savings, superior access to real-time data for analysis and decision-making, superior inventory management, and more.

Let's examine these propositions in detail, and the technology that makes them possible. The new EDI is already emerging as the driving force behind the use of XML on the Web.

7.1.4 *Ubiquitous EDI: A quantum leap forward*

Ubiquitous EDI will have a profound impact on business-to-business and business-to-consumer relationships. The many problems with current implementations of EDI have relegated it to large enterprises and selected industries. However, the combination of the Internet, Internet-based technologies, and XML will open up EDI not only to small-to-medium enterprises (*SMEs*), but also to individuals (Example 7-1).

Through deployment of these technologies, EDI will experience growth and market penetration that will rival the e-mail market. Electronic commerce will finally blossom on the Web and become an everyday part of our lives. In short, EDI will usher in a new era in computing. The Internet will metamorphose from a transport for Web pages into a ubiquitous and seamless foundation for every imaginable transaction. In the future, EDI will touch every aspect of computing.

Various forms of "data interchange" have been implemented with various degrees of success. Examples include OLE and DDE for sharing data among heterogeneous applications on the same computer. CORBA, Java RMI, COM and COM+ are generalized technologies for data interchange

©1998 THE XML HANDBOOK™

Example 7-1. The value of data interchange.

Mike opens his company expense report, and in the microsecond it takes to launch, he reminisces about the old days when he had to fill out these things himself. Now the computer does it for him. Mike recently took a trip to Utah to close a major deal. In the process he purchased a plane ticket, a rental car and various meals. In the old days, he used to enter all of these charges manually into an expense program...not any more.

Mike uses a corporate American Express card for these purchases. When he opens the expense report, it automatically connects to American Express, via EDI, and presents a list of new charges. Mike selects the charges that are appropriate for this expense report.

American Express sends this data to Mike's computer, which automatically formats the data into his expense report. Mike then clicks the send button and the expense report is sent to his manager to approve. Then the company's bank instantly wires the money to Mike's bank account.

Behind the scenes, all these companies are establishing connections, as needed, to share information in a secure and reliable manner using XML and the Internet. But Mike doesn't concern himself with what goes on behind the scenes, he's off to close another big deal in Washington.

among systems. Then there is the traditional EDI market for the "Electronic exchange of data to support business transactions".[1]

In focusing on the traditional EDI market, the seminal questions are: "What is the real value of EDI?" and "Why should I care?"

7.1.5 *The value of EDI*

While traditional EDI is very costly and difficult to implement, the potential benefits are very significant. Companies that have implemented EDI rave about benefits like improved efficiency, vendor management, cost savings, superior access to information for decision making, tighter inventory control, customer responsiveness, and its a competitive advantage that can be marketed to attract new customers.

1. *European Workshop on Open Systems Technical Guide on Electronic Commerce (EWOS ETG 066)*

EDI was initially implemented to improve efficiency by enabling companies to eliminate costly and slow manual methodologies, like the processing of purchase orders and bills. It was thought that by allowing the computers of two or more companies to share this information, they could achieve dramatic improvements in efficiency.

However, the largest savings are derived from a complete shift to EDI that allows companies to completely eliminate their hardcopy processes. The traditional 80/20 rule applies in reverse to EDI, meaning that it is the last 20% of your trading partners to convert to EDI who account for 80% of the potential savings.

This is because even with 80% of your trading partners using EDI, you must still maintain the same manual processes for the remaining 20% who don't. While most companies have not been able to completely convert from hardcopy processes to EDI, the 20% savings companies have realized have still been very significant. With ubiquitous EDI enabling companies to completely eliminate their manual processes, the savings will improve dramatically.

With EDI, companies are also able to manage their supply chains much more efficiently. Through EDI, companies have been able to reduce the average time from issuance of an order to receipt of goods from several weeks, to a matter of days. By improving inventory control, companies are able to minimize their investment in costly inventory, while still being able to address spikes in business. For industries where inventory costs are a significant part of their business, like manufacturing, this represents a significant cost savings.

EDI also reduces the financial float by eliminating the typical order generation, delivery and processing, by 5-7 days. By combining EDI with Electronic Funds Transfer (EFT) companies can also reduce the financial float by 8-10+ days. Based on the amount of money[1] involved, this can represent a significant savings.

EDI also provides companies with superior real-time information upon which to base decisions. Everyone recalls stories of companies who simply didn't have the data to realize how bad things were, until it was too late. With EDI, companies have access to complete data in real-time. The ability

1. *Electronic Commerce/Electronic Data Interchange and Electronic Funds Transfer (EC/EDI/EFT)*, `http://www.dfas.mil/dir_init/ec_edi/index.htm`

to collect, manipulate and measure information about your relationships with vendors and customers can be critical to your company's success.

Customer responsiveness is becoming increasingly important. Many companies have leveraged technology to dramatically improve customer responsiveness. A good example of this is Federal Express, which has created a Web site where customers can track the status of their packages.

This is only accomplished through FedEx's end-to-end dedication to EDI. By capturing information about the package status at each step in the process, and making this information accessible to customers, they have made themselves leaders in customer support. This is critical to building and growing businesses, especially in the Internet-age.

Some companies who have implemented EDI with one supplier, have gone on to market this capability to other potential customers, as a unique selling point. This has enabled them to grow their business. As EDI becomes more ubiquitous, the tide could shift to the point where companies will not accept vendors who are not EDI-capable. This refers back to the dramatic savings that can be achieved by a complete conversion to EDI.

7.2 | Traditional EDI: Built on outdated principles

"Traditional" EDI is based on outdated principles that will cause it to fade into technological obscurity, as it becomes embraced and replaced by the "New" EDI. Traditional EDI refers to the use of rigid transaction sets with business rules embedded in them. This model simply does not work in today's rapidly changing business environment.

This problem is compounded by the fact that companies have chosen to interpret these transaction set standards in ways that suit their unique business requirements. As a result, vendors who engage in EDI with multiple customers typically must create a unique solution to handle the transaction sets from each company. This makes the implementation of EDI far too expensive, especially for SMEs.

These and other problems have hindered the growth of EDI. However, by solving the problems of traditional EDI, we will usher in a new era, where EDI is as common as an Internet account is today.

©1998 THE XML HANDBOOK™

7.2.1 *The history of EDI*

EDI is a process for exchanging data in electronic format between heterogeneous applications and/or platforms in a manner that can be processed without manual intervention.

EDI dates back to the 1970s, when it was introduced by the Transportation Data Coordinating Committee (TDCC). The TDCC created transaction sets for vendors to [1] follow in order to enable electronic processing of purchase orders and bills.

At the time, the technology landscape was very different from what it is today. Lacking ubiquitous powerful CPUs, a common transport, and a file format that allows for flexibility, they defined strict transaction sets. These transaction sets addressed the needs for data content, structure and the process for handling the data. In other words, the business rules were embedded into the transaction set.

The incorporation of business rules into the definition of the transaction set causes many problems, because:

1. Business rules vary from company to company;
2. Business rules for one size company may be completely inappropriate for companies of another size;
3. Business rules are subject to change over time according to changes in market dynamics.

In short, the use of fixed and rigid transaction sets, while necessary at the time, have limited the value of EDI, and therefore stunted its growth.

7.2.2 *EDI technology basics*

Traditional EDI is based on fixed transaction sets. These transaction sets are defined by standards bodies such as the United Nations Standard Messages Directory for Electronic Data Interchange for Administration, Commerce and Transport (EDIFACT), and the American National Standards Institute's (ANSI) Accredited Standards Committee X12 sub-group.

1. *The Mythical Value of EDI Standards* by Alan Chute, `http://www.filex.com/filex/edimyth.htm`

Transaction sets define the fields, the order of these fields, and the length of the fields. Along with these transaction sets are business rules, which in the lexicon of the EDI folks are referred to as "implementation guidelines".

To actually implement EDI, the trading partners would follow these steps:

1. Trading partners enter into an agreement, called a trading arrangement.
2. They select a Value Added Network (VAN).
3. The trading partners typically either contract for, or build themselves, custom software that maps between the two data set formats used by these trading partners.
4. Each time a new trading partner is added, new software would have to be written to translate the sender's data set for the recipient. In other words, you start from scratch with each new trading partner.

Transaction sets are typically transmitted over expensive proprietary network service providers called VANs, which generally base charges on a mixture of fixed fees and message lengths. These fees can become quite substantial, but they are typically overshadowed by the cost to build and maintain the translation software. The VANs provide value-added services such as:

1. Data validation (compliance) and conversion
2. Logging for audit trails
3. Customer support
4. A secure and stable network
5. Accountability
6. Transaction roll-back to support uncommitted transactions

It is important to note that EDI is not simply the exportation of data from one system to another, but a bi-directional mechanism for interaction between systems. Because these disparate systems typically employ different file formats (data notations), schemas, data exchange protocols, etc., the process of exchanging data is very difficult.

7.2.3 *The problems of traditional EDI*

Traditional EDI suffers from many problems that have limited its growth. One of the most significant problems is the fact that it is based on the transfer of fixed transaction sets. This rigidity makes it extremely difficult to deal with the normal evolution necessary for companies to introduce new products and services, or evolve or replace their computer systems.

In addition, these transaction sets include strict processes for handling the data. These processes are not universally acceptable to companies in various industries and of various sizes. This problem is compounded by a standardization process that is too slow to accommodate the accelerating pace of business today.

In addition, the high fixed costs of implementation have been too much to justify for SMEs. In short, there are a host of problems which, despite the benefits of EDI, have prevented its universal adoption.

7.2.3.1 Fixed transaction sets

EDI is currently built on transaction sets that are fixed in nature. For example, a contact field might include the individual's name, title, company, company address and phone number. However, the company does not have the flexibility to add or subtract fields.

Why is this important?

Companies cannot be frozen in time by a fixed transaction set. This prevents them from evolving by adding new services or products, changing their computer systems and improving business processes. This inflexibility inherent in the current custom solutions required to map data between each trading partner pair is untenable, despite the significant benefits of EDI (Example 7-2).

7.2.3.2 Slow standards evolution

EDI standards are defined by standards bodies that are structurally ill-equipped to keep up with the rapid pace of change in the various business environments they impact, as illustrated by Example 7-2.

These standards accommodate many companies with very different needs. They also encompass not just the ontology, but the associated busi-

©1998 THE XML HANDBOOK™

Example 7-2. Problems of traditional EDI: Healthcare

The transaction sets created for the healthcare system were defined for the traditional indemnity model, where the insurance company pays the doctor on a per visit basis. However, the movement toward managed care was not foreseen in this transaction set. Since managed care pays the doctor a set fee per patient, but does not reimburse on a per visit basis, the standard transaction set simply doesn't work.

The typical doctor sees a mixture of patients, some having managed care insurance and others with indemnity insurance. In order to accommodate this scenario, the doctor is forced to create a false "per visit" fee for managed care patients. This false fee, which is required in order to "complete" the transaction set, creates havoc with the doctor's other billing systems, which EDI was supposed to help.

Rigid transaction sets that enforce process as well as content are simply not flexible enough to address the ever-changing business environment.

ness processes. As a result, it is very slow and difficult, if not impossible, to develop one-size-fits-all solutions.

The current process for defining standards for transaction sets can take years. This simply will not work in today's business environment, which is characterized by accelerated change and increased competition. However, in an effort to jump-start the creation of industry ontologies in the form of DTDs for XML, the work of the traditional EDI standards bodies could be enormously valuable.

Historically, technology standards that are defined and managed in a top-down fashion, like EDI standards, have been replaced by bottom-up standards that allow for independent and distributed development. In other words, technologies like XML, that support greater flexibility and diversity, while providing compatibility between implementations, typically replace inflexible managed solutions like fixed transaction sets. The XML standardization process is managed by the World Wide Web Consortium (W3C).

7.2.3.3 Non-standard standards

Despite the perception of standardization, there remains some flexibility in the interpretation of these standards. The simple fact of the matter is that

companies have unique needs, and these needs must be translated into the information they share with their trading partners.

In practical terms, the customer is at a significant economic advantage in defining these "standards", vis-a-vis the supplier. As a result, suppliers are forced to implement one-off solutions for each trading partner. In many of the industries where EDI is more prevalent, the suppliers also tend to be the smaller of the two partners, which makes the financial proposition even worse (see 7.2.3.4, "High fixed costs", on page 108).

Because of the various informational needs of companies, it is impractical to expect that EDI standards can be a one-size-fits-all proposition. The variables of company size, focus, industry, systems, etc. will continue to create needs that are unique to each company. As evidence, consider the amounts companies spend on custom development and customization of packaged applications.

7.2.3.4 High fixed costs

While large companies tout the financial and operational benefits of EDI, these same benefits have eluded the SMEs. That is because of the high fixed costs of implementation, which must be balanced against savings that are variable.

Depending on the level of automation, implementing EDI for a large enterprise is not substantially more expensive than it is for SMEs. In fact, it can be more expensive for the SMEs. Larger companies can often implement a single EDI standard, while the SMEs must accommodate the various standards of their larger partners. This can be very expensive.

Yet, ironically, the benefits are variable. So, if savings are 2% of processing costs, this might not be a substantial number for the manufacturer of car seat springs, but it can be a huge number for GM, Ford or Chrysler. SMEs simply do not have the scale to compensate for the high fixed costs of traditional EDI.

Because of this some of the SMEs that claim to implement EDI are actually printing a hardcopy of the data feeds and re-typing them in their systems. The reason they implemented this faux-EDI is to meet customer requirements, but they simply do not have the transactional scale to justify the investment. Something must be done to bring down these costs (Example 7-3).

Example 7-3. Problems of traditional EDI: Retail

One large retailer requires its vendors to implement EDI in order to qualify as a vendor. However, like all traditional EDI implementations, the data set is unique to the retailer.

For small companies, implementing this system can be quite an investment. Retail is a very fast-paced industry, because it is forced to cater to ever-changing customer demands. As a result, some suppliers to this retailer have implemented this costly technology, only to later lose their contract with the retailer. In fact, because of the significant investment in technology these companies were forced to make, they have sued the retailer.

If this technology were universally applicable, the vendor's investment in a single customer would be eliminated, as would the retailer's legal liability.

7.2.3.5 Fixed business rules

Business rules are encapsulated in the definition of the transaction sets as implementation guidelines. However, business rules are not something that can be legislated, nor can they be rigid.

Business rules that are applicable for a large enterprise, may be completely inappropriate for an SME. To make matters worse, business rules for a medium-sized enterprise may be wholly inappropriate for a small enterprise.

These business rules will also vary between industries. Even companies that are in the same industry, and the same size will implement different business rules. What's more, business rules change over time. The earlier healthcare example demonstrates this point.

Traditional EDI focuses too much on process as an integral part of the transaction set. This is a fatal flaw. New technologies, like XML, support the separation of process, or business rules, from the content and structure of the data. Achieving this separation is critical to widespread adoption of EDI.

The linkage between transaction sets and business rules creates additional problems. The real-life implementation of EDI typically requires custom solutions for each trading partner pair. This creates havoc when trying to implement or modify global business rules.

For example, if your company changed business policy to begin accepting purchase orders, which you had refused to accommodate in the past, you would have to manually change the individual software for each trad-

©1998 THE XML HANDBOOK™

ing partner. You could not make these changes on a global basis using traditional EDI.

This problem also impacts your ability to upgrade or replace your internal systems, since they are uniquely woven into the EDI software in place. In essence, you can become locked into systems that may become obsolete by the time you actually implement the total solution.

7.2.3.6 Limited penetration

EDI penetration has been very limited, when compared to the penetration rates of other automation technologies. Yet the majority of the value of EDI is derived by complete elimination of the hard-copy processes EDI is meant to replace.

As mentioned above, EDI benefits do not follow the 80/20 rule, because converting the first 80% of your vendors to EDI results in only 20% of the potential cost savings. The remaining 80% of the costs remain, since the company is forced to maintain all of the old manual process in tandem with the electronic processes. The most significant savings come only from completely replacing all manual processes with EDI.

The real value of any network is in its adoption by users. Remember Metcalfe's Law: The value of any network is roughly proportional to the number of users squared.

But EDI, in its current state, is *not* a single interlinked network. On the contrary it is a series of one-to-one chains of data flow. As a result, it is vulnerable to alternative "networked" solutions like those enabled by XML, the Internet, Internet-based services, and database connectivity.

7.3 | The new EDI: Leveraging XML and the Internet

Now that we've established the tremendous benefits of EDI, and the structural problems of traditional EDI, the obvious question is: "How can we fix the problems?"

Fortunately, new technologies are coming together to completely reshape the EDI landscape. Today, EDI is currently implemented in a 1-to-1 man-

ner between trading partners. These partnerships can then be extended through tiers to create a supply chain.

This is all changing!

The new paradigm is the *supply web*. The supply web is based on utilization of XML, the Internet, Internet-based services and database connectivity to create a network, or "web", of trading partners.

Implementation and operational costs will plummet, trading partners will implement one-size-fits-all solutions, and adoption will skyrocket And the benefits will not be limited to the trading partners, they will be driven down to end-users as well. EDI will become as commonplace as e-mail.

In short, EDI will dramatically alter the business computing landscape, moving the world forward from our current islands of automation toward a single fabric of commerce tying together businesses and end-users.

Traditional EDI is based on the technologies that existed in the 1970s. Now it is time to build a new EDI architecture on current technologies like XML, the Internet, Internet-based services and database connectivity.

- XML provides the ability to separate the data and structure from the processes.
- The Internet provides the ubiquitous connectivity upon which a Web of interconnected trading partners can flourish.
- Internet technologies provide a layer of security, authentication, transactional support and more, to support the needs of EDI.
- Database connectivity means that XML data, and the business rules that interact with that data, can be communicated among disparate systems by means of middle-tier data filters and aggregators.

Together, these technologies will remove the barriers to widespread adoption of EDI. By leveraging these technologies, EDI will become more flexible, more powerful, less expensive and ultimately ubiquitous.

7.3.1 *XML*

XML is closely related to HyperText Markup Language (HTML), the original document representation of the World Wide Web, as both are based on SGML. While HTML enables the creation of Web pages that can be

©1998 THE XML HANDBOOK™

viewed on any browser, XML adds tags to data so that it can be processed by any application. These tags describe, in a standardized syntax, what the data is, so that the applications can understand its meaning and how to process it Example 7-4.

For example, in HTML a product name and a price might be somewhere in the text. But the computer only knows that there is a collection of characters and numbers. It cannot discern that this data represents a product name and price. As a result, little can be done with the data.

With XML, however, the product name is tagged (e.g. product_name), as is the product price (e.g. product_price). More importantly, there is an association between the product price and the product name.

This information results in significant additional value. For example, a user can now search for the best price on a specific product.

Example 7-4. XML insulates applications from diversity: Customer record

The following example demonstrates one of the values of XML. Below are three different types of message documents from three different companies (A,B and C). Each describes its respective company's customer data:

Company A:

```
<Person name="Mike Hogan" phone="6502864640"
                    E-mail="mph@poet.com" />
```

Company B:

```
<Person name="Mike Hogan" street address="999 Baker Way"
       city="San Mateo" zip="94404" phone="6502864640"/>
```

Company C:

```
<Person name="Mike Hogan" phone="6502864640"/>
```

The XML parser parses, or disassembles, the messages to show the "person" element, which has associated attributes ("name", "phone", etc.). These attributes, as you can see, differ in content and organization.

However, if your application was written to extract a person's name and phone number, it could work equally well with each of these document types without modification. In fact, if these companies evolve their data to include additional information, your application continues to function without modification. This flexibility is one of the benefits of XML.

XML documents must be "well-formed", which means that most document-type information – grammar and hierarchy – can be embedded in the tags that "mark up" the individual document. There can also be an associ-

ated *document type definition* (DTD), containing additional meta-information that describes the data.

In either case, XML is self-describing. As a result, applications can be very flexible in their ability to receive, parse and process very diverse sets of information. This enables companies to write a single application that will work with diverse sets of customers. In fact, such a system is even capable of processing information from new trading partners in an ad hoc fashion. This capability completely changes the dynamics of EDI.

Using XML, companies can separate the business rules from the content and structure of the data. By focusing on exchanging data content and structure, the trading partners are free to implement their own business rules, which can be quite distinct from one another. Yet, using templates, companies can work with legacy EDI, non-XML datatypes as well.[1]

7.3.2 *The Internet*

Many companies heralded the cost savings and ubiquity of the Internet as the death knell for VANs. However, this future has not come to pass...yet.

The boldest of these claims was based on the notion that the extranet would redefine the new computing paradigm. What these pundits failed to realize was that the Internet alone does not address the needs of the EDI community.

The EDI community is generally limited to the largest enterprises. EDI is mission critical, and requires a dependable network. It also requires a level of security that couldn't find on the Internet. To put it simply, the savings were not sufficient to justify the switch.

Furthermore, connectivity is only a small part of the problem, the largest issue is the exchange of data in a universal fashion.

All these issues have now been addressed.

- Technology is now available to provide dial-up services to support the Internet in addressing up-time and throughput for mission critical information.
- Security has improved dramatically.

1. *Introducing XML/EDI*, `http://www.geocities.com/WallStreet/Floor/5815/start.htm`

- The use of XML will broaden the EDI customer base to include SMEs and individuals. This new group of customers is much more price sensitive, so they are inclined to seek an Internet-based solution.
- The ability to exchange data in a more democratic and ad hoc manner will cause an explosion in the average number of EDI connections.

The current average number of EDI trading partners, for those companies who utilize EDI at all, is two. Building EDI solutions based on XML, and operating this over the Internet, which offers a low-cost ubiquitous transport, will dramatically expand the value of EDI, according to Metcalfe's Law.

7.3.3 *Internet technologies*

Internet technologies have improved, and continue to improve dramatically, now providing a critical mass of technologies that is capable of replacing the services of VANs. Consider the following list of VAN services, each followed by the Internet-based alternatives that offer greater functionality and flexibility:

Data validation and conversion

XML DTDs, XML validation, templates, and structure-based data feed interpretation.

Intermediary-based logging for audit trails

Ubiquitous XML-savvy repositories employed by all trading partners enables rich logging for audit trails. Combining these with electronic signatures ensures system and company identification.

Consulting, customer service and customer support

This function could be handled by VANs capable of making the transition to Internet technologies, or by the other legions of consultants.

Security and accountability

Public key cryptography, certificate authorities, digital signatures can assure secure transactions.

Connection reliability, stability

New technologies in bandwidth allocation, general improvement in the stability of the Internet and alternative fail-over solutions like dial-up continue to move the Internet toward supporting critical real-time data flow. (Remember, it was originally designed to withstand nuclear attack!)

Trading partner negotiation

Directories (X.500, LDAP, NDS, Active Directory), certificate authorities, digital signatures, e-mail, Internet versions of the Better Business Bureau, etc., can support this function.

Transactional support (roll-back, etc.)

The improvements in remote messaging systems and transaction processing monitors provide a layer of transaction support that is capable of adding transactional integrity even on unstable networks.

Because of the knowledge and experience of the VAN community, and because of the anticipated growth of the entire EDI market, the VAN community is well positioned to transition into consulting or systems integrator roles, helping companies implement these new technologies.

7.3.4 *XML data storage*

In other technological transitions, data storage has been a moot point, since the data could be mapped more-or-less directly into relational tables or file systems. More recently, object-oriented database management systems became available for this purpose.

XML data, however, is composed of self-describing information elements that are richly linked, and that utilize a hierarchical structure and naming mechanism. These qualities enable new data-access capabilities based on the tree structure, such as context-sensitive queries, navigation, and traversal.

©1998 THE XML HANDBOOK™

"Native" XML-based support for these new capabilities can be provided by a value-added content management layer above the DBMS. *POET CMS* is an example of an object-oriented data storage solution that was designed for this type of use.

7.3.5 *Data filtering*

The source of the vast majority of EDI-related information is currently in mainframes and relational databases. This data will be marked-up on the fly with XML tags. XML data will also come from data sources such as:

- XML content management systems
- Various Internet resources
- EDI-XML documents, both full documents like purchase orders and short inter-process messages
- Result sets from applications, also in XML

These diverse sources must be communicated with by a middle-tier "data filter" that can speak to each source in a manner that the source will recognize. The data must then be filtered in source-dependent ways, based on one's confidence in the data, application of consistent business logic, resolution of the various element-type name ontologies, response mechanisms, security, caching for performance, etc. Only then can the application address the data in a consistent manner and receive consistent responses from the middle tier.

The role of a data filter is shown in Figure 7-1. Products such as *POET Object Server* and *POET CMS* can be used to build one.

The middle tier could maintain valuable meta-information that would add structure and context to the data stream. Such information could include:

- Routing for the query, response, etc.
- Source of the information (to indicate credibility, etc.)
- Time stamps
- Data, DTD, and tag normalization
- Context and navigation aids

©1998 THE XML HANDBOOK™

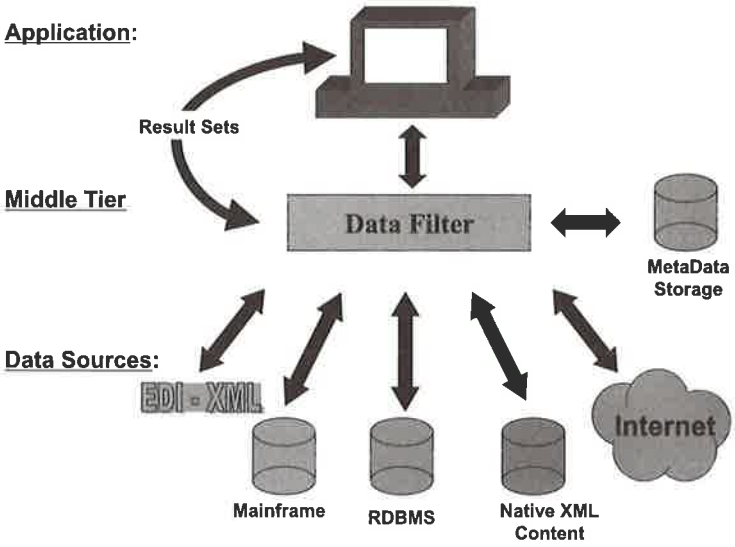


Figure 7-1 Data filter in an XML-based EDI system

Further details on XML content management and the use of object-oriented storage systems in the management of XML data can be found in Chapter 27, "POET Content Management Suite", on page 364.

7.4 | Conclusion

After decades of investment in corporate data centers, we have created islands of automation inside companies. Their isolation from trading partners limits the value companies can recognize from these systems.

EDI offers the ability to change all of this. EDI offers benefits like:

- improved efficiency
- supply chain management
- real-time data and metrics
- better planning
- superior execution
- control systems
- resource management

- cost savings
- superior access to information for decision making
- customer responsiveness
- ... and more.

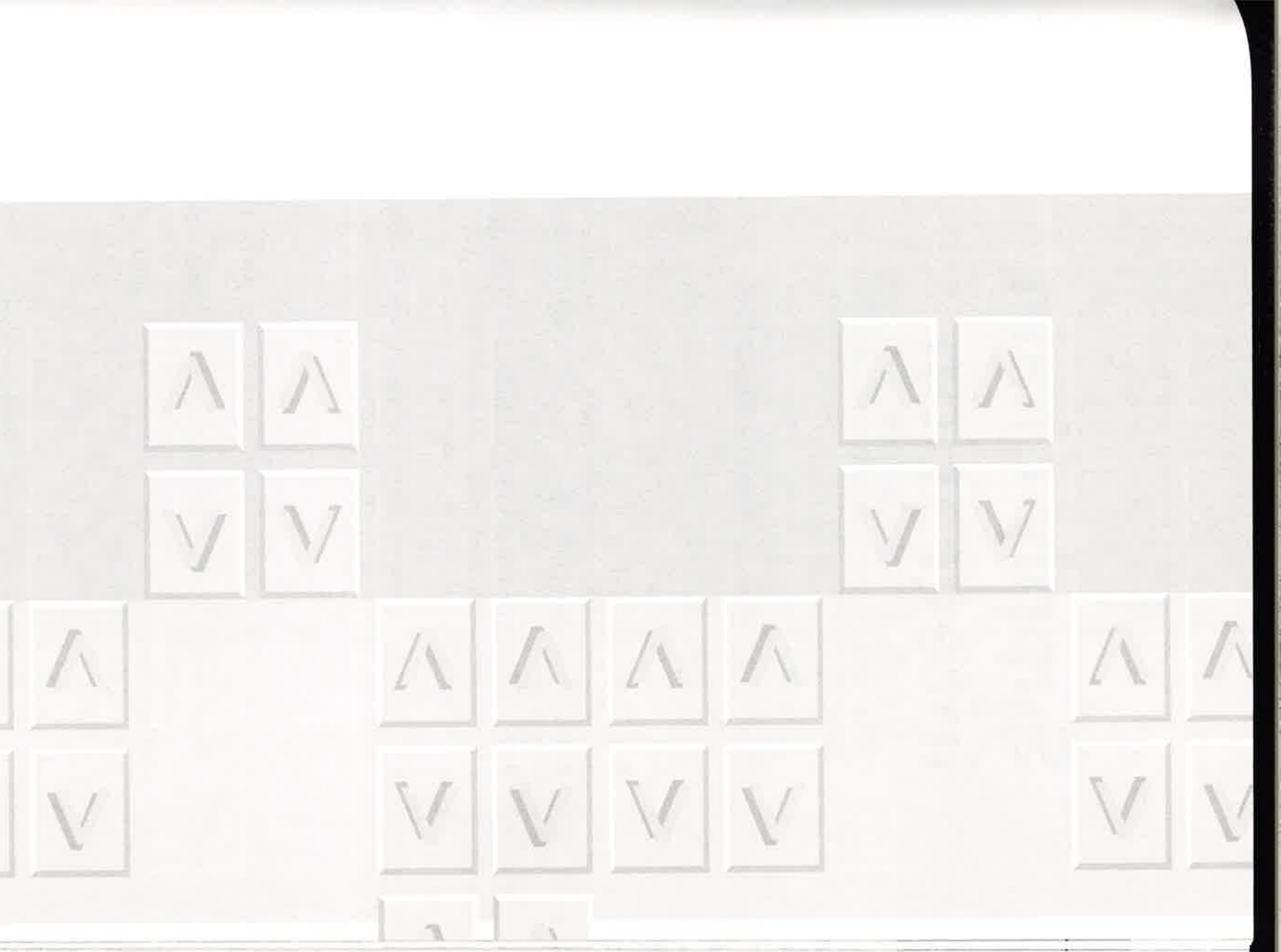
However, traditional EDI is very difficult and expensive to implement. Because of problems like rigid transaction sets that embed business rules, slow standards development, high fixed costs, and limited market penetration, EDI has not achieved broad adoption. Fortunately, new technology is now available to address these problems and, in the process, reshape the EDI industry.

XML, the Internet, Internet-based services and database connectivity are combining to create a revolution in EDI. Instead of forcing companies to adapt their systems and business processes to the EDI data, this data will dynamically adapt to the companies' existing systems.

EDI will no longer be isolated to certain industries or the largest enterprises, it will become as ubiquitous as e-mail. EDI will transition from a one-to-one supply chain to a richly interconnected web of trading partners forming the supply web. Proprietary networks will become extinct, over time, and VANs will evolve or evaporate.

This supply web will result in dramatic improvements in efficiency. Companies will slash costs, while improving access to critical information. This information will be pushed all the way to the end-user, providing superior customer support as well.

©1998 THE XML HANDBOOK™



Securities regulation filings

- EDGAR: The U.S. Securities and Exchange Commission quarterly reporting system
- Document creation and revision under rigid conformance requirements
- Software visualization assistance for authors

©1998 THE XML HANDBOOK™

Supply chain integration

- Middle-tier Web application
- Web Interface Definition Language (WIDL)
- XML Remote Procedure Call (RPC)

©1998 THE XML HANDBOOK™

Chapter

8

Supply chain management is tricky business for suppliers and manufacturers. Ideally, they should have real-time access to one another's inventory and schedule planning systems, but that is expensive to achieve with traditional EDI. webMethods, `http://www.webmethods.com`, who sponsor this chapter, and Joe Lapp, who prepared it, believe there is a better means of business-to-business communication.

Managing a manufacturing business is in some ways a lot like feeding a family. The home is the factory and grocery stores are the suppliers. Parents manufacture meals for themselves and for their children.

But before they can make a meal they must be sure the refrigerator is stocked with the right foods in the necessary quantities. As they serve dinners, the food levels diminish, so they have to time their shopping to be sure that they always have enough food for the next meal. Sometimes the family must scale up to feed unexpected in-laws, and sometimes the family must scale back when little Billy's big belly is staying at Grandma's.

8.1 | Linking up a supply chain

Manufacturers and suppliers struggle with these issues all the time. Suppliers are caught up in the challenge, as they must feed parts to multiple manufacturers. They must be sure that manufacturers have parts when they're needed, but they have to be careful not to overstock their products when

manufacturer demand is not high enough to sell them. The supplier may itself be a manufacturer and have its own suppliers.

A series of businesses that feed parts to one another in sequence is known as a *supply chain.*

Suppose Manufacturer X decides to maximize the efficiency of its link in a supply chain. It wants to integrate its Manufacturer Resource Planning (MRP) system with the planning systems of its suppliers, thereby providing each side with rich inventory information in real-time.

The manufacturer benefits by having access to up-to-date availability information of supplier parts, including parts delivery schedules. The supplier benefits by having access to the manufacturer's current parts inventory levels and to the manufacturer's expected rates of depleting the inventory.

8.2 | Supply chain integration requirements

Manufacturer X does not want to lose an arm or a leg or a bevy of shareholders in the process of implementing the solution. It requires an inexpensive solution that it could put together in a period of weeks, rather than in a period of months or years. This rules out traditional Electronic Data Interchange (EDI).

In an earlier time, EDI might have won hearts and ears and pocketbooks, but now the Internet and XML offer better ways to do things. Manufacturer X is aware of what is currently possibly with technology and imposed the following general requirements:

- The system must integrate with Manufacturer X's existing MRP system.
- Manufacturer X must communicate with its suppliers over the Internet. Private network solutions are too costly.
- Access to manufacturer data must be secure. Only registered suppliers may access the data. Suppliers may not access the data of other suppliers. Data must remain secure in transit over the Internet.
- The effort and expense required of both Manufacturer X and its suppliers must be minimized.

The problem can be solved by employing a tool that allows disparate applications to interoperate over the Web. Let's take a look at one.

8.3 | The *B2B Integration Server*

The webMethods *Business-to-Business Integration Server* (B2B) sits between applications to enable them to communicate despite differences between them. The applications need only agree in an abstract sense on the nature of the services they offer, and on the data to be exchanged between these services.

The server employs *WIDL* (Web Interface Definition Language) technology for expressing these abstractions. Once the abstractions have been established, any two applications can communicate, regardless of their programming languages, whether they accept and/or receive XML messages, and regardless of the DTDs to which the XML messages conform.

In other words, B2B makes applications accessible to one another over the Web, and it makes existing Web data accessible to applications. It provides the communications infrastructure needed to do the job, including security, passage through firewalls, and access to proxies. It also translates between message representations, such as URIs, CGI query data, and differing XML message document types.

Let's see how these capabilities can be applied to integrate a supply chain.

8.4 | Overview of the system

A somewhat generalized version of the system architecture is depicted in Figure 8-1.

The generalization allows us to demonstrate the applicability of the solution to supply chains in general. Here, the B2B server sits on the manufacturer's site and mediates all exchanges between suppliers and the manufacturer. B2B assumes responsibility for hiding network, protocol, and security issues from the supplier and manufacturer systems, and hiding differences in how the systems are interfaced.

Supplier systems access the manufacturer's MRP system to obtain part inventory levels, and communication between the systems is completely

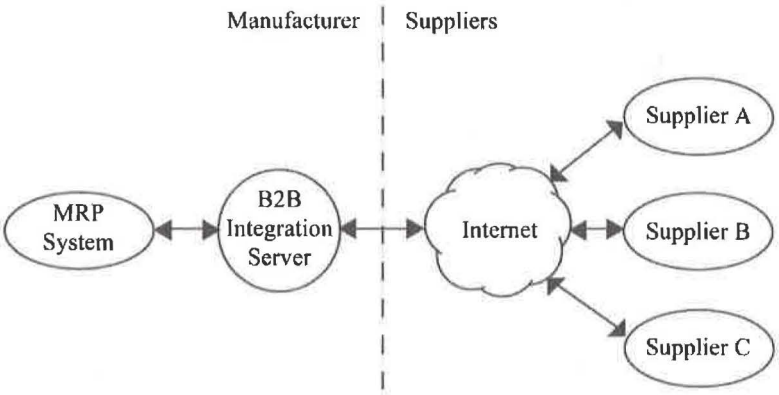


Figure 8-1 Supply chain integration architecture.

automated. The suppliers issue requests to the B2B integration server in the form of XML messages, sending the requests via the standard POST method of HTTP.

B2B translates these requests into calls to the MRP system. It then translates responses from the MRP system into XML reply messages that it sends back to the supplier. This request/reply mechanism for accessing services is called *Remote Procedure Call* (RPC).

The MRP system must also access supplier data. In order to minimize the impact on the suppliers, B2B uses standard URIs and CGI queries for the requests and allows both HTML and XML to be used in the responses. (See Chapter 38, "WIDL and XML RPC", on page 554 for more information on these technologies.)

Upon receiving a response, the B2B server uses WIDL to convert the HTML or XML into a data representation that is suitable for the MRP system to consume. It then passes the converted data to the system, completing the request/response circuit.

8.5 | The manufacturer services

The manufacturer services comprise half of the complete integration solution. These services give suppliers access to inventory level information found in the manufacturer's MRP system.

Figure 8-2 shows how the manufacturer provides services to its suppliers. A supplier issues a request to the B2B server, which in turn calls upon a piece of software known as a "plug-in." The plug-in acts on the MRP system to perform the request and returns data back to B2B. B2B translates the data into the appropriate form for delivery to the supplier.

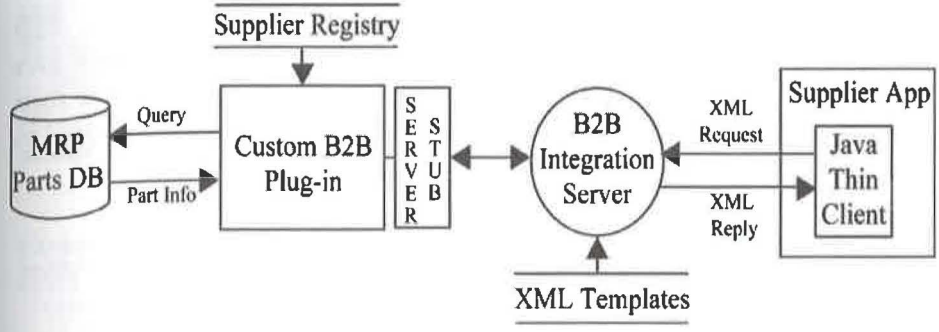


Figure 8-2 Providing suppliers with manufacturer services.

Several pieces shown in Figure 8-2 are key to the solution and merit some discussion.

8.5.1 *B2B plug-in*

The plug-in is code that Manufacturer X wrote to communicate with its MRP system. It is written in Java and exposes an interface to the B2B server. The most important "method" (or program function) of this interface is one that retrieves part information from the MRP system when the method is called. The method inputs a part number and outputs information about the part.

The plug-in will only return part information to the suppliers that provide the parts; suppliers cannot acquire information about the parts that other suppliers provide. The plug-in accomplishes this by looking up the supplier's user name in the supplier registry to fetch the associated supplier ID. The retrieved ID must match the supplier ID that the MRP database associates with the part.

8.5.2 *Server stub*

A server stub is a portion of code that links into the plug-in and that allows the B2B server to invoke an API that the plug-in exposes. Server stubs enable the B2B server to communicate with plug-ins written in any programming language. They also benefit the server by hiding the details of the plug-in's method signatures (that is, its name and parameter definitions).

When a supplier requests B2B to invoke a manufacturer service, B2B hands the input parameters to the server stub, telling the stub which Java method to invoke on the plug-in. The stub invokes the method and then provides B2B with the method's output parameters, which B2B returns to the supplier.

8.5.3 *XML requests and replies*

B2B communicates with the supplier via XML. It receives XML requests from the supplier and it sends XML replies back to the supplier. When B2B receives a request it translates the XML into set of input parameters and hands these parameters to the server stub. When the stub returns output parameters, it translates the output parameters into an XML reply.

Manufacturer X chose to represent these XML request and reply messages using a "generic" message DTD. A generic DTD is capable of representing any set of input or output parameters, thereby allowing all message exchanges to use the same DTD. For efficiency, the solution uses an encoder/decoder module to translate XML into input parameters and to translate output parameters into XML.

8.5.4 *Java thin client*

The Java thin client is a piece of software that Manufacturer X developed and distributed to all of its suppliers. It contains the webMethods thin client, which allows the client to submit and receive XML messages. However, the supplier could choose to use any XML-aware client.

The thin client provides suppliers with default behavior to jump-start their integration efforts with software that understands the generic XML DTD. To use the manufacturer services, the thin client must first establish a

secure SSL session and log in to the server with a user name and a password that Manufacturer X provided.

8.5.5 *Manufacturer interface specification*

The solution requires that we define the set of services that the stub offers, and it requires that we state the data inputs and outputs for each service. We accomplish this by using WIDL 3.0 to define an interface specification.

Example 8-1 shows a portion of the interface specification that does the job. A supplier invokes the "getInventory" method to retrieve inventory information as a function of a part number.

Example 8-1. WIDL interface specification for the manufacturer services.

```
<WIDL NAME="com.Manufact-X.PartsInventory" VERSION="3.0">
  <RECORD NAME="PartHandle">
    <VALUE NAME="partNumber"/>
  </RECORD>
  <RECORD NAME="PartInventory">
    <VALUE NAME="inventoryLevel" TYPE="i4"/>
    <VALUE NAME="targetLevel" TYPE="i4"/>
    ...
  </RECORD>
  ...
  <METHOD NAME="getInventory" INPUT="PartHandle"
      OUTPUT="PartInventory"/>
  ...
</WIDL>
```

The developer uses the B2B release of the *webMethods Automation Toolkit* to generate the server stub from this interface specification. The toolkit includes GUI-based tools for designing the interface specification and for generating the source code for the stubs, so that the developer does not need to be familiar with either WIDL or XML.

8.6 | The supplier services

The supplier services comprise the second half of the complete integration solution. These services give the manufacturer access to supplier inventory levels and delivery schedules.

Figure 8-3 portrays how the manufacturer utilizes the services of the supplier. Suppliers make their information available from Web servers in the form of HTML or XML pages.

8.6.1 *Client stub*

The client stub provides APIs that the plug-in calls to access the information found on these pages. The plug-in runs a background thread that periodically invokes these APIs to retrieve supplier part information. The thread updates the MRP system with the part information that the APIs return.

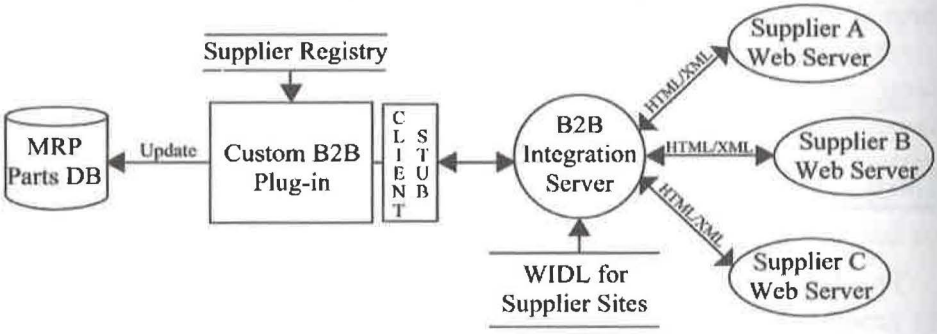


Figure 8-3 Providing the manufacturer with supplier services.

The B2B server uses Web automation to make the supplier Web sites available to the plug-in. B2B can provide Web automation services to any application, not just plug-ins, but Manufacturer X wanted to centralize the entire integration solution within the plug-in.

Web automation *wraps a Web site* so that it looks like a set of APIs (functions). As shown in Figure 8-3, there is no need to put Web automation technology on any of the wrapped web-sites themselves. The B2B server

merely sits between the Web site and the client (in this case, the plug-in) and makes the web-site accessible to the client through the APIs of a client stub.

8.6.2 *Supplier interface specification*

A developer generates the client stub by first designing a WIDL 3.0 interface specification for the supplier services. Example 8-2 shows a portion of this specification.

The interface specification defines the APIs that the stub will expose, including the input and output parameters of each API. Since the plug-in is written in Java, the stub implements the APIs as Java methods. The developer links the client stub into the plug-in so that the plug-in can call these methods. The stub methods in turn use the services of the B2B server.

Example 8-2. WIDL interface specification for the supplier services.

```
<WIDL NAME="com.Supplier.PartAvailability" VERSION="3.0">
  <RECORD NAME="LoginProfile">
    <VALUE NAME="username"/>
    <VALUE NAME="password"/>
  </RECORD>
  <RECORD NAME="Availability">
    <VALUE NAME="dateRefreshed"/>
    <RECORDREF NAME="parts" DIM="1" RECORD="Part"/>
  </RECORD>
  <RECORD NAME="Part">
    <VALUE NAME="partNumber"/>
    <VALUE NAME="availableInventory" TYPE="i4"/>
    <VALUE NAME="quantityInTransit" TYPE="i4"/>
    ...
  </RECORD>
  ...
  <METHOD NAME="login" INPUT="LoginProfile"/>
  <METHOD NAME="getAvailability" OUTPUT="Availability"/>
  ...
</WIDL>
```

Next the developer uses WIDL to wrap the supplier Web sites so that each site conforms to the interface specification. The Toolkit does this through direct interaction with a Web site, and again the developer need not have any knowledge of WIDL. Once the WIDL files have been created,

one configures the B2B server to wrap the Web sites by dropping the files into a directory.

Each supplier site then has the same interface, consisting of the set of methods that the client stub exposes. The input parameters of a method fill out a form on a supplier Web site. The output parameters of the method contain data extracted from the pages that the site returns upon submitting the form parameters.

This approach allows the suppliers to return the information in any form – HTML pages using any presentation or XML messages using any DTD. The supplier can even have a B2B Integration Server receive the form parameters and reply with XML messages, allowing the supplier to have tight integration with other manufacturers as well. The manufacturer's B2B server makes the form of the supplier data transparent to the plug-in.

Once per day, the plug-in iterates over the suppliers listed in the supplier registry. For each supplier it retrieves a supplier ID, a URI, a user name, and a password. The URI specifies the location of the supplier site. The user name and password are items that the supplier provided to the manufacturer; they allow the manufacturer to log in to the supplier site.

For each supplier in the registry, the plug-in invokes methods on the client stub. It first invokes "login" to authenticate with the supplier's site and then invokes "getAvailability" to acquire the part availability data. Finally, the plug-in writes the part availability data to the manufacturer's MRP database, keeping the database accurate to within a day.

8.7 | Conclusion

The B2B Integration Server allows Manufacturer X to integrate tightly with its suppliers. It allows the manufacturer's MRP system to communicate with the supplier planning systems without requiring the MRP system to have any knowledge of the Internet, of XML, or of the supplier system interfaces.

These facilities enabled Manufacturer X to implement the solution in only two weeks. Had Manufacturer X gone with a CORBA solution or with a solution involving traditional EDI, it would still be negotiating the platform and protocol details with the suppliers. The *B2B Integration Server* provided Manufacturer X with a significantly simpler, faster, and less expensive way to get the job done.

Comparison shopping service Web site

- Middle-tier Web application
- Virtual Database technology (VDB)
- Dynamic merchant Web site aggregation

©1998 THE XML HANDBOOK™

Chapter 9

Comparison shopping is a universal pastime, and it has never been easier than on the World Wide Web. Junglee Corporation, http://www.junglee.com, sponsors of this chapter, have created a technology that makes it easier still, and a working Web site to prove it. The chapter was prepared by Anand Rajaraman, STS Prasad, and Debra Knodel.

Despite the fears of some publishers that the Web would dry up the market for books, so far the reverse has been true. Not only are people buying books about the Web and its technologies, but book selling over the Web is widely regarded as proof of the viability of electronic commerce.

9.1 | Shopping online for books

But there is plenty of room for improvement in the process. Consider what the experience is like today.

A shopper seeking the best price for the best books on a given topic doesn't have an easy time of it today. The shopper must view several Web sites, initiate a search on each site for the desired product, and interpret the results of each search.

Unfortunately, every Web site has a separate structure and vocabulary for search and results presentation, making it awkward for the shopper to quickly evaluate the results. Figure 9-1, for example, shows the difference

between the search and result pages of two online bookstores, Amazon.com and Powells.com.

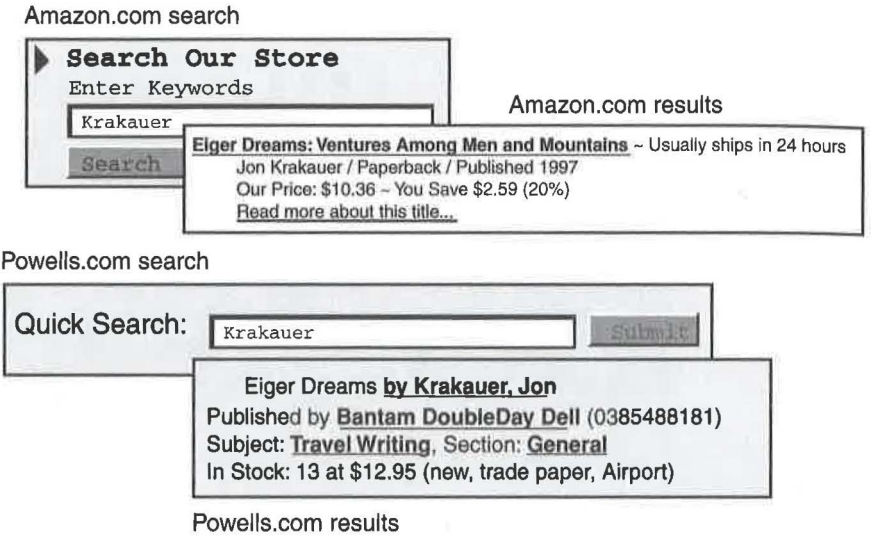


Figure 9-1 Two bookstores, two views.

It would certainly be more convenient if all bookstores could be searched with a single query, and all the results could be presented together. There is now a Web service that does just that.

9.2 | The *Junglee Shopping Guide*

The *Junglee Shopping Guide* is a configurable comparison shopping service that uses *Virtual Database* (VDB) technology to aggregate dozens of merchant Web sites across a wide range of product categories, consolidating them into a single shopping guide.

A customer enters specific attributes about a desired item, such as brand name, price range, author (for books) or category (for gifts). The *Shopping Guide* does the rest, automatically searching through thousands of products from a variety of merchants.

The results are displayed in a single table. From it the customer can make informed purchase decisions, comparing product features, availability, and price.

The *Shopping Guide* leverages the power of XML to deliver results in a form that can be manipulated by the browser without round trips to the Web server. For example, XML allows browser-side sorting and filtering of data, and presentation of the data to suit specific user preferences based on stylesheets.

VDB technology makes this possible by transforming the Internet into a database. Data is collected from Web sites based on a user query, and structured into a standard representation for each category. The search results are delivered in XML, using a uniform document type definition (DTD) for all sites, regardless of the original form of the data (typically HTML).

Figure 9-2 shows a sample result from book shopping.

```
<?XML version"1.0"?>
<!DOCTYPE booklist SYSTEM "book.dtd">
<booklist>
<Book>
  <Merchant>Amazon.com </Merchant>
  <Title> Eiger Dreams </title>
  <Author> Krakauer, Jon </Author>
  <Format> Paperback </Format>
  <Price> $10.36 </Price>
 <Availability> Ships in</Availability>
</book>
</booklist>
```

Figure 9-2 Sample books document.

Let's look at the underlying technology.

9.3 | How the *Shopping Guide* works

Figure 9-3 shows how the VDB manages this process using wrappers.

©1998 THE XML HANDBOOK™

A *wrapper* is a Java program designed to extract data from Web sites. The wrapper may in turn use *extractors* (not shown) to extract attributes from unstructured (non-XML) text. Data transformations and data validations are applied to determine data integrity.

A wrapper is created for each Web site, while an extractor is typically created for an entire collection of Web sites with similar information. An extractor consists of extraction rules and dictionaries to provide sophisticated linguistic processing for unstructured text

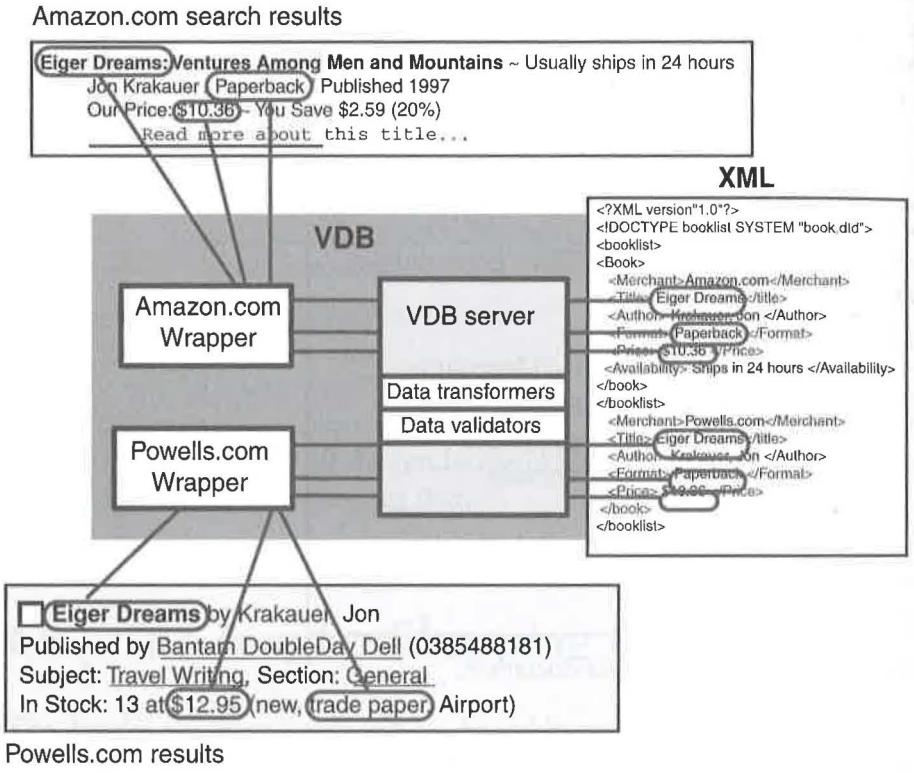


Figure 9-3 How the VDB works.

Once the data is gathered and transformed, it can be presented in response to a query in a combined form, such as the one shown in Figure 9-4. The results are displayed using the XML DSO in *Internet Explorer 4.0.*[1]

Dynamic HTML and data binding are used to create a compelling user experience. The Web shopper can choose between multiple views and can manipulate the data from the browser without the need for round trips to the server.

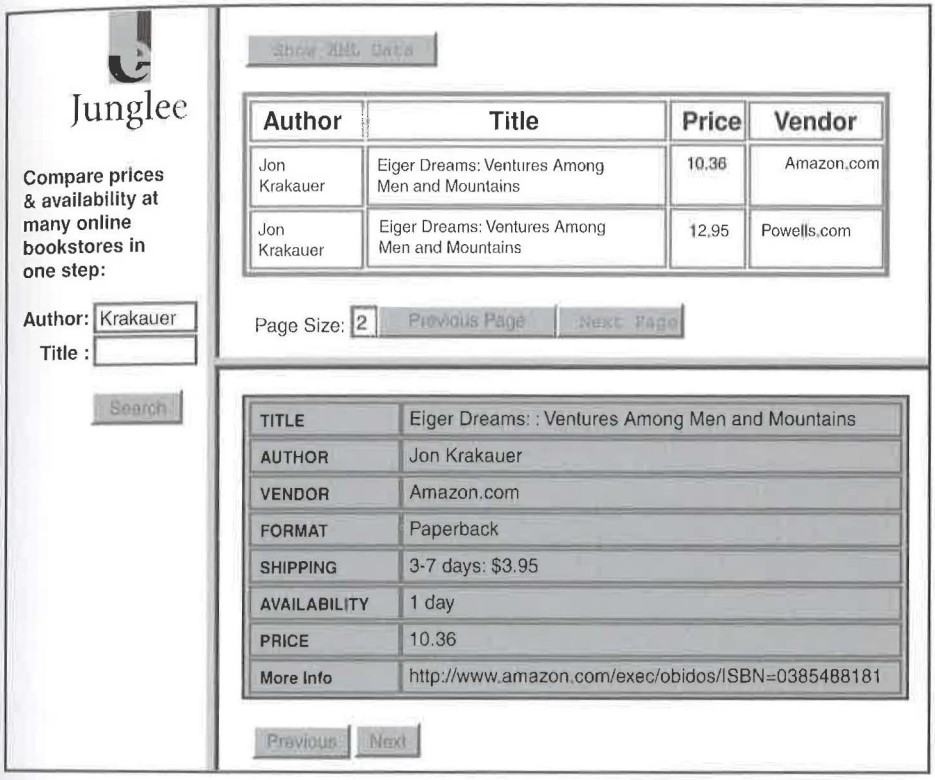


Figure 9-4 *Shopping Guide* unified results in XML.

9.4 | Conclusion

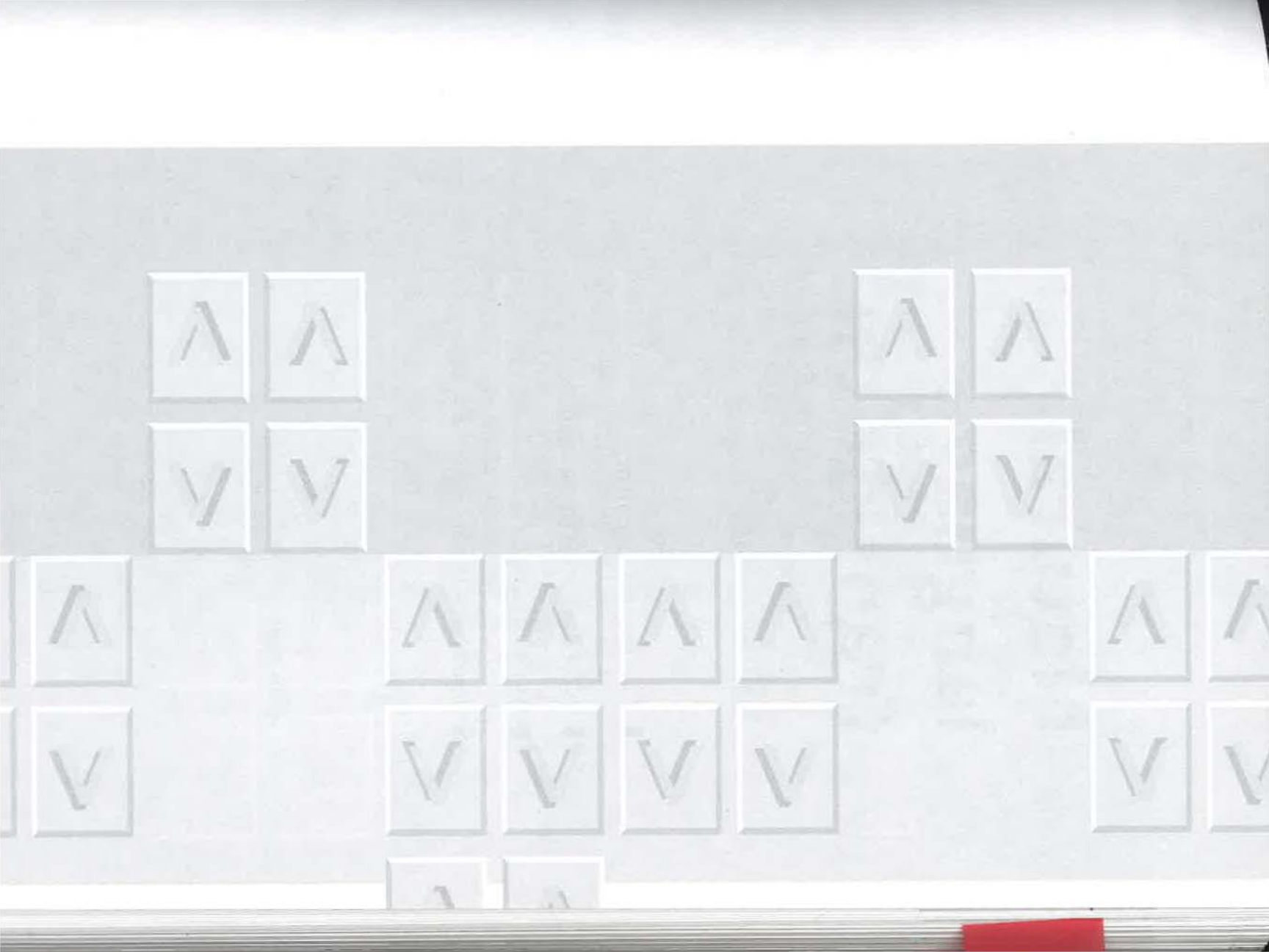
VDB technology transforms the Internet into a database enabling powerful structured searches, while XML provides an efficient mechanism for deliv-

1. A *Netscape* version is being developed.

ering structured data to browsers. You can learn more about VDB in Chapter 29, "Junglee Virtual DBMS", on page 386.

The XML version of the Junglee *Shopping Guide* is a working prototype that illustrates the power of XML and VDB technology. It provides access to hundreds of merchants and millions of products.

Tip *You can experience the XML Junglee Shopping Guide in action at* `http://www.junglee.com/tech/xml_demo.html`. *Until browser support for XML is ubiquitous, a version that presents its results in HTML is available at* `http://www.junglee.com/shop/index.html`



Natural language translation

- Content management
- New directions in translation
- Versioning
- Reuse

©1998 THE XML HANDBOOK™

Chapter

10

Translating documents can be a real challenge in today's global market, but content management can improve the process dramatically. Chrystal Software, a Xerox New Enterprise company, `http://www.chrystal.com`, sponsored this chapter to show you how. It was prepared by Robin Gellerman, Steve Kiser, and Simon Nicholson.

Companies have tremendous opportunities to reach new customers in today's global market. Improved global communications allow companies to be known instantly around the world. Modernized manufacturing practices make it possible to create a diverse set of products brought to market faster than ever before. Fewer trade barriers allow products to be sold and distributed globally.

10.1 | Mistakes can be costly

In the rush to new markets, some make careless mistakes that can be costly. Here are a few examples from the Marketing Hall of Shame:

- Parker Pen planned to introduce the ball-point pen in a new country with the slogan: *"It won't leak in your pocket and embarrass you"*. The company introduced its product with the mistranslated slogan: *"It won't leak in your pocket and make you*

pregnant". This was not the message that Parker Pen originally conceived.

- The sales figures for the Chevy Nova in several Spanish speaking countries were far below expectations. After some analysis, GM found a tremendous problem. It wasn't in the car itself, but in the name. Nova means "no go" in Spanish – not a characteristic many of us hope for when buying a car.
- After repeated requests from their biggest customers in five countries, a telecommunications manufacturer quickly added a security menu option to a new model of cellular phone. Although the new menu item was a trivial engineering change and was implemented within a day, the documentation could not keep pace. The change to the English, French, German, and Spanish versions of the documentation took six weeks - thirty times longer than the product change.

These examples, whether truth or urban myth, illustrate the importance of accurate and timely language translation in today's global marketplace. These companies experienced short-term embarrassment for their mistakes. Other mistakes could be far more costly. What if the process of translation was so complicated that it slowed the introduction of your product to a new market by a year? What if a mistranslation introduced a safety issue for the consumer?

10.2 | It's a small world

Products are coming to market faster, compressing product delivery times. At the same time, expansion into new markets increases the need for more languages. In response to these pressures, companies must accelerate time tables, streamline processes, and manage document creation and translation concurrently. This is not a luxury – it is a requirement.

Opening of global markets is not the sole reason for this increased need for multilingual documents. Other factors are at play. The creation of international organizations through partnerships and mergers such as Rover and BMW, Ford and Mazda, and the European Union are increasing the demand. Employees are more mobile and more geographically dispersed. More source document content is written in languages other than English.

Although English is recognized as the international language of business, there remain several compelling reasons for delivering information in the native language of the consumer. First, it allows you to more effectively communicate information and ideas to the target audience. Second, in some cases, documentation in the local language is not just a good idea-it is the law. Companies may not be able to carry on business unless they provide localized documentation. And for many, translation is required just to keep up with the competition.

The more companies recognize the strategic benefits of concurrent product development and document delivery, the more serious the need for solutions becomes. XML and content management are providing those solutions today. And, as a result, making the world a much smaller marketplace.

This chapter outlines the process used to translate technical information, describes the challenges and areas for improvement, and explains how XML and content management help companies control costs and make production more efficient.

10.3 | Business challenges

10.3.1 *Cost containment*

In the technical documentation market, companies spend millions of dollars annually on translation. An IBM study in 1995 estimated $50 billion was spent for translation worldwide with an estimated annual growth rate of 15%. And the cost is growing, driven by the desire for simultaneous worldwide launch of new products and the need to support new languages and markets.

In some cases the development of documents in a different language variant can be as high as 10% of the cost of the development of the source. With global translation spending forecast to grow at 15% per annum, businesses need to implement new solutions now.

Translation is expensive, costing 15 to 25 cents per word including revisions. It is also time-consuming: Translation and proofreading with one review cycle takes many hours per page or chapter. Additional revisions cost even more time and money.

©1998 THE XML HANDBOOK™

10.3.2 *Fast-paced product development*

In most industries, product development schedules are now a fraction of what they were a decade ago. Also, products in many industries demand a frequent "technology refresh" to stay competitive, increasing the frequency of product versions. With as little as six weeks to deliver a product, these companies cannot afford to double that time to revise and translate documentation. These compressed schedules force companies to establish concurrent processes.

Rapid product development also allows companies to manufacture more makes and models of their products. This increases complexity of multiple document variations that must all be released simultaneously. Managing the slight variations between products and their supporting documentation is the key to streamlining the translation process.

10.3.3 *Diverse documents*

The diversity of documents that must be developed, delivered, and maintained is growing. Figure 10-1 shows an example.

An automotive manufacturer makes a design change to a part that is used in five different car models. All of the accompanying documentation, including the workshop manuals and owners guide must be modified to reflect the change.

Even if that change results in modifying only two paragraphs of text, each medium used to distribute information, such as print, CD-ROM and the World Wide Web, must be updated. Now multiply that by the number of languages needed in each country where the cars are sold. It is apparent that this is a sizable problem.

10.4 | Translations today

The greatest productivity improvements to be realized in the translation process are not in the translation itself, but in making the entire process more efficient. Although technology exists for automatic machine translation, it is not perfect. Machine translation lacks the subtlety and error

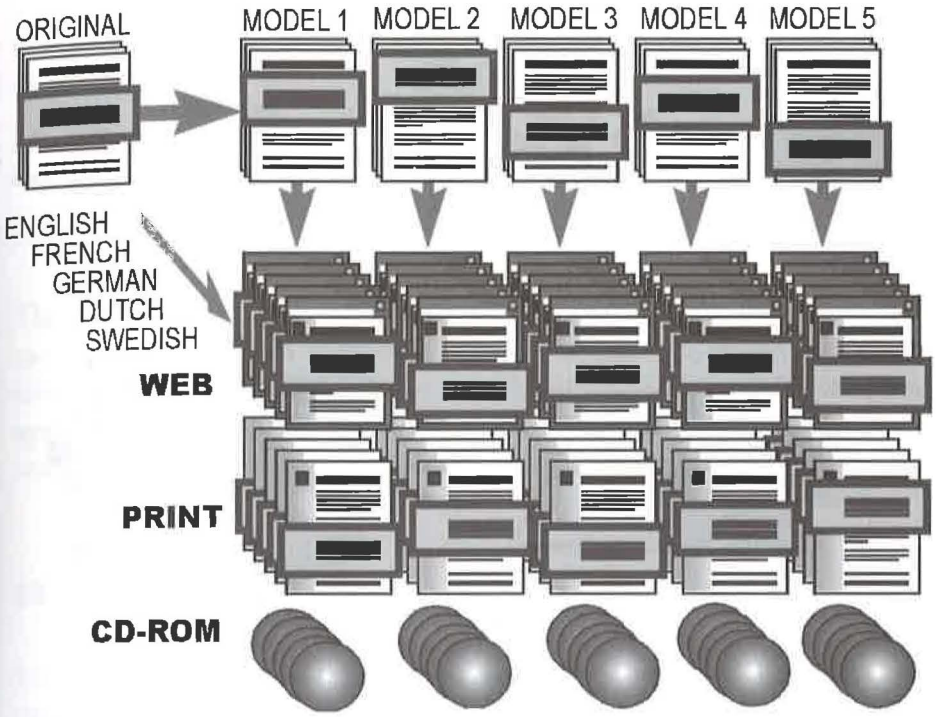


Figure 10-1 Complexity is multiple products, languages, and media.

checking provided by a human translator. Translation remains an art, not an exact science.

Here is how a typical document gets revised and translated today (Figure 10-2). Generally, the work begins on the source language document. A variety of people including photographers, authors, technicians, marketing and legal, create and review the text and graphics. After identifying the changes and redundant content, the material is sent out for translation. This manual process reduces unnecessary translation and allows the translator to avoid re-translating content that has not changed. The newly translated material is returned and integrated into a document called a "variant", a document containing the translated content for a single language. Since most companies require multiple languages, there will be multiple variants that must be kept in sync with the source.

As changes occur to the source document, incremental revisions are sent out for translation. During this stage, translators may need to directly inter-

©1998 THE XML HANDBOOK™

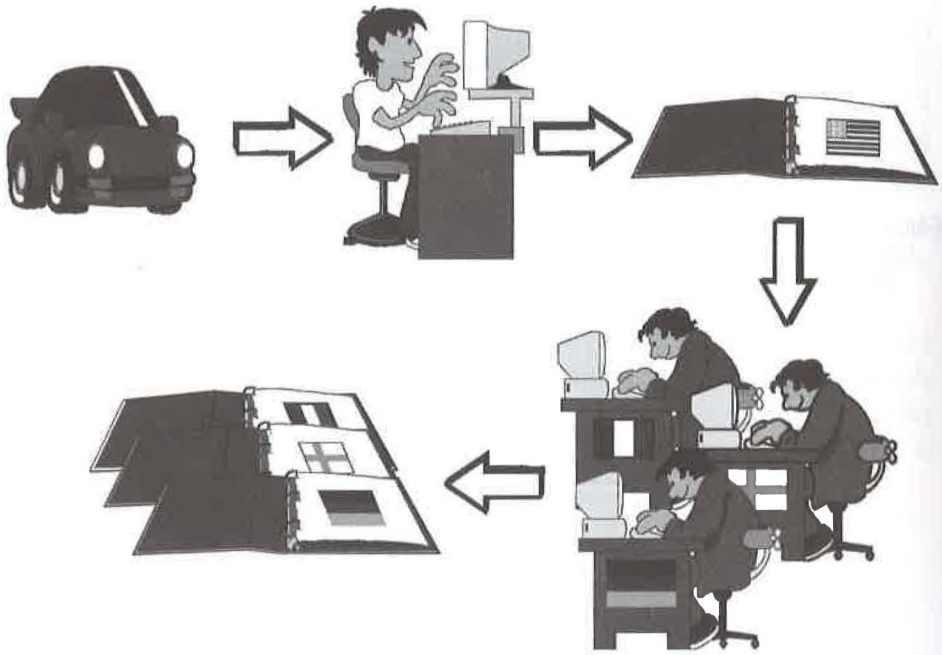


Figure 10-2 The process of translating documents.

act with the original author to resolve ambiguities in meaning which are bound to exist.

Things really heat up when product development is complete. The final content, often including changes to previously submitted content, is sent out for translation. The final set of translations are then integrated into the variant documents. After translation to the target languages is completed, a final check ensures all changes are complete, and reflected in the translations and inserted in the proper places.

Translation is a specialized service, so most companies contract with experts to perform this task. The most accurate translators are native speakers, often requiring pieces of the document to be sent all over the world. This increases the complexity of communication and causes synchronization nightmares when a last-minute engineering change occurs. And you thought it was hard to communicate with your documentation department down the hall!

Last minute changes can put on-time delivery at risk, leaving you three choices to meet the delivery date: Increase the translation staff, extend the

delivery date, or ship the product without appropriate documentation. Given the huge cost, both monetarily and in customer satisfaction, which would you choose?

10.5 | New directions

The biggest challenge in the management of today's language translation is to manually keep track of content - what is new, reused, revised, translated, reviewed, and approved. With the combination of XML and content management, companies can automate this critical function to simplify processes, shorten time-to-market, reduce rework, and control costs.

10.5.1 *Components*

The key to making the translation process more efficient is to work with smaller parts. XML breaks up the information into smaller information components (Figure 10-3). The smaller and more specific the component is, the more addressable and reusable it is. With smaller information units, it is easier to pinpoint changes, translate only new information, and automatically update information reused throughout the document. Similar to advanced technologies and methods used in engineering and manufacturing of items such as cellular phones, new cars, and software, components simplify complexity and increase flexibility for adapting to change.

A component is a piece of information that can be used independently, such as a paragraph, chapter, instructional procedure, warning note, part number, order quantity, graphic, side-bar story, video clip, or one of an infinite variety of additional information types.[1] For the translation process, components have a positive and profound impact.

When managed by a content management system, these components can be controlled, revised, reused, and assembled into new documents.

1. For more information about components, see Chapter 26, "Astoria: Flexible content management", on page 352.

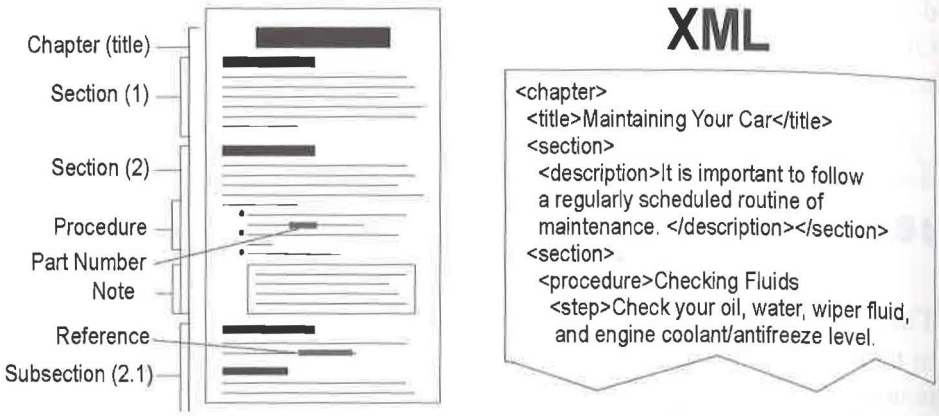


Figure 10-3 Document components described with XML.

10.5.2 *Reduce reinvention with reuse*

The ability to reuse components within documents has an important impact on the translation process. Reuse can be as simple as locating a component from one document and linking it into a new document. This method of linked reuse instead of copying makes updates more efficient. When components stored in a content management system change, the information is revised only one time. All of the documents containing that component are automatically updated.

Reuse is helpful even before translation. Using a content management system, the original technical writer can create a standard glossary containing translated terms and phrases (Figure 10-5). With assistance from the content management system from inside the authoring tool, the writer is prompted with approved terms, phrases, and other content.

The writer can easily reuse components or create new terms. This allows for greater control over document content and terminology reducing ambiguities, inconsistencies, and unneeded rework. One example of a controlled language is Simplified English, used in the aerospace industry.

The type of automated reuse shown in Figure 10-4 will also help in the translation process. The glossary can be translated once into the target languages. The glossary terms will be automatically inserted into the language prototype before being passed to the translator. This reduces the volume of work by reducing the amount of new material.

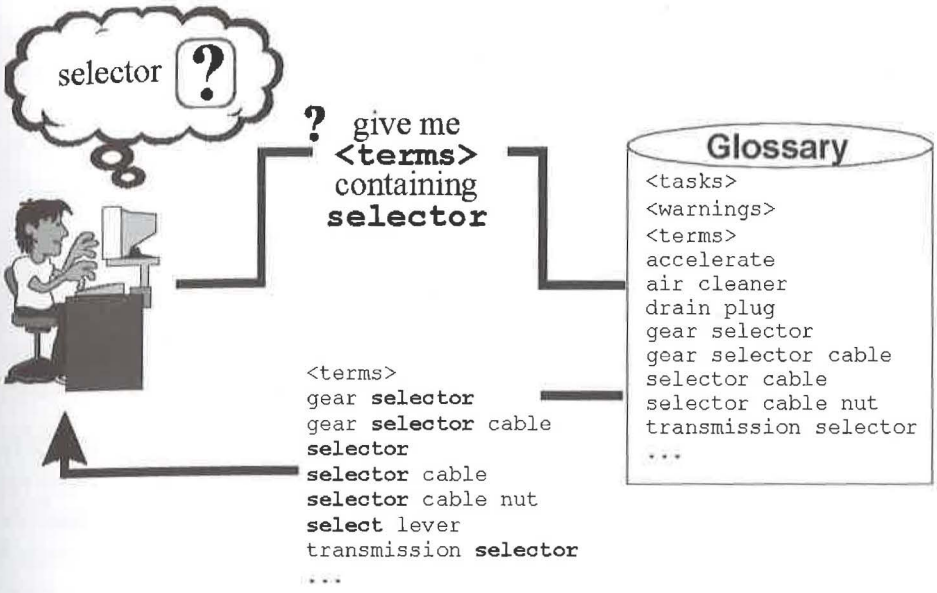


Figure 10-4 Reuse of glossary terms reduces rework.

When a standard term in the glossary needs to change, the update is made once. All the documents using the term are automatically updated. Reuse makes it simpler to identify changes and reduces rework. This approach can reduce total translation time by 50% and reduce costs by 15%.

10.5.3 *Identify changes with versioning*

When components are maintained in a content management system, changes can be automatically recorded and tracked (Figure 10-5). Without burdening the author, the system automatically collects revision information and identifies what changed. Instead of sending the entire document, only modified components need to be extracted and sent for translation. This streamlines the process because it reduces the volume of data being sent out and relieves the translator from manually identifying the changes from previous versions.

Another benefit of components is that they can be translated as they become available rather than waiting for a complete document. This makes it possible to manage concurrent translation along with product development.

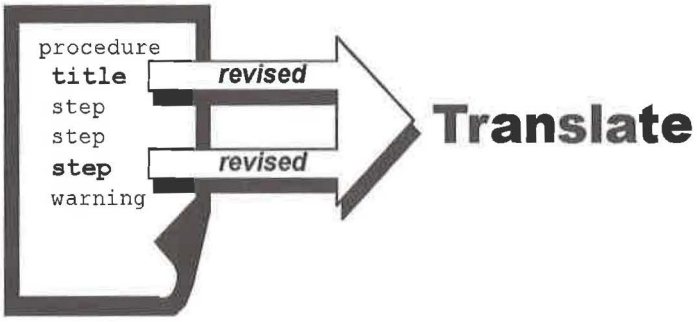


Figure 10-5 Component versioning saves time by identifying only pieces that change.

10.5.4 *Alignment enables concurrent authoring and translation*

When stored in the content management system, the source and target language of the document are guaranteed to be aligned with identical structures (Figure 10-6). For instance, if the source contains a <procedure> element with a unique attribute of "123," the language variants will also contain this element with this attribute.

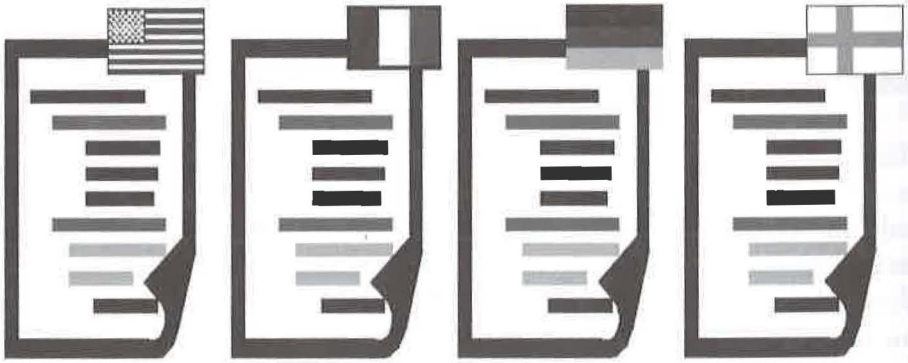


Figure 10-6 Source and target language documents have identical structures.

This alignment has several immediate advantages. First, when a source language component is ready for translation, its language variants are also identified. These components can be clearly identified, locked, and submitted to the translator. After translation, the components are correctly versioned, providing a full life-cycle history of all components including the language variants. Since the structures are identical, it is also easy to identify and distribute changes to end-users, regardless of language used.

10.6 | In the real world

A major automotive manufacturer produces all the documentation required to deliver a fully serviceable automobile to buyers. This documentation includes in-car handbooks, workshop manuals, customer representative training guides and reference material. The documentation is translated into a variety of languages for the company's global markets. Currently, these guides are delivered in a hard copy format as well as in multimedia such as CD-ROMs.

After installing Chrystal Software's Astoria content management solution and implementing concepts described in this chapter, composition time for each language version went from about three weeks to less than 2 days. With the time saved, the company can better manage the translation process. They now review documents in-house and control updates and versions within the content management system, reducing the dependency on translation suppliers for review. As a result, overall production time for updates was cut by 50%.

By using XML and content management for managing the creation, translation, and revision of multiple document sets, companies can:

- Reduce initial translation costs by reusing common content across documents.
- Improve document consistency through controlled vocabulary authoring.
- Improve re-translation by pinpointing re-translation units.
- Shorten time to market by overlapping authoring and translation processes and minimizing volume of translation.

©1998 THE XML HANDBOOK™

Chapter 11

Filing documents to conform to government regulations is a necessity in enterprises throughout the world. Although knowing that the misery is shared won't make the task any easier, the information in this chapter can. The chapter is sponsored by Interleaf, Inc., `http://www.interleaf.com`.

Government documents tend to be rigid in their requirements and must be completed with extreme care. The penalties for error can be large. Many enterprises have seized on XML (and its parent, SGML) as a solution. An XML DTD and validating parser can enforce many of the requirements for completing a government filing.

Indeed, many government agencies use XML themselves and some allow or even require their filings to be XML documents. However, even if XML use isn't a filing requirement, there are plenty of benefits from using XML to prepare your filings, as we'll see in this chapter.

But creating XML in the raw can be intimidating to the non-programmer business analysts or other content experts who must prepare the filings. These professionals could take charge of their own XML documents if they had the right sort of visual tools.

11.1 | Visualizing an XML document

The following examples illustrate the difference between writing XML element type declarations and generating them with a visual tool. Figure 11-1 shows a DTD in the standardized plain-text form that is processed by XML software. To create this, the actual XML element type declarations must be typed out, completely and correctly.

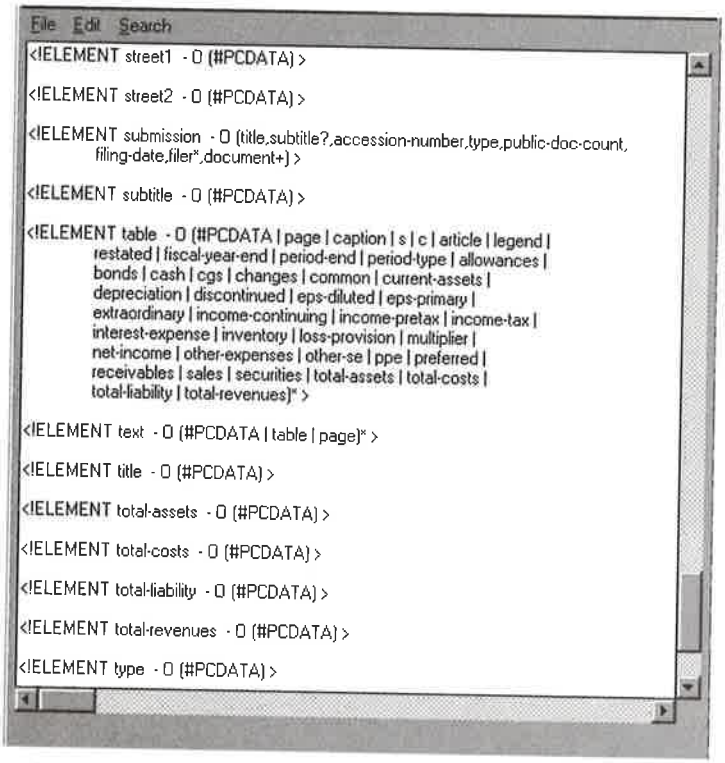


Figure 11-1 Standard plain-text XML DTD.

Alternatively, a DTD could be assembled using a visual tool (see Figure 11-2). The user need only point and click the mouse on the appropriate element type (in this case, `business-address` was selected), and the tool automatically generates the opening and closing tags. The tool also shows all the element types that are subordinate to `business-address`, such as `street1` and `street2`. Additionally, when defining the DTD for

©1998 THE XML HANDBOOK™

business-address, the content developer can indicate which related element types are mandatory, thereby ensuring that all the necessary elements are completed.

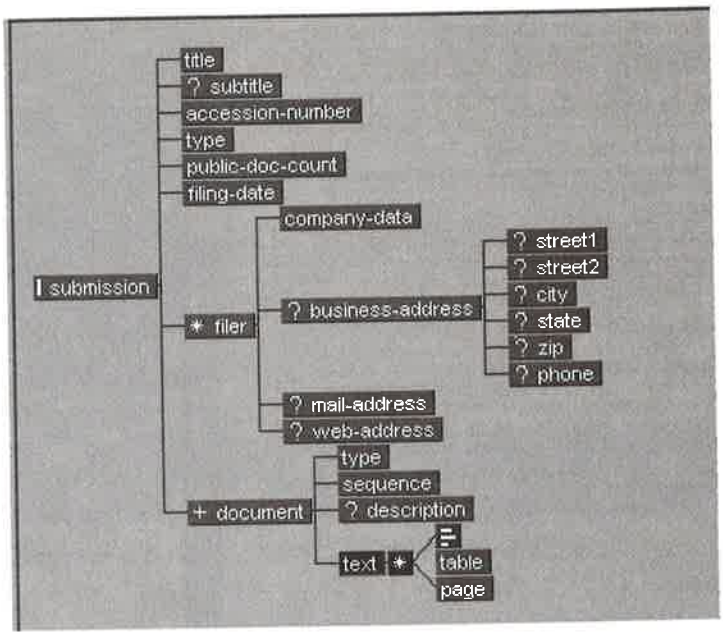


Figure 11-2 Visual DTD modeling.

In Figure 11-3, a document is being created for the U.S. Securities and Exchange Commission (SEC). It is a quarterly report known, for some reason, as *EDGAR*.

The data and markup are entered manually in a plain-text programmer's editor. There are no built-in validation checks to ensure that the markup is correct, with the correct spelling, syntax, and closing tags (although some of those could be validated externally). And there are no simple means for aligning columns when editing tables.

Figure 11-4 shows what is possible with a visual tool. In the screen shot, the items on the left strip (outside the document margins) identify the XML elements on each line. And the user can insert the XML tags graphically into the document to improve the appearance and the alignment of the table columns.

©1998 THE XML HANDBOOK™

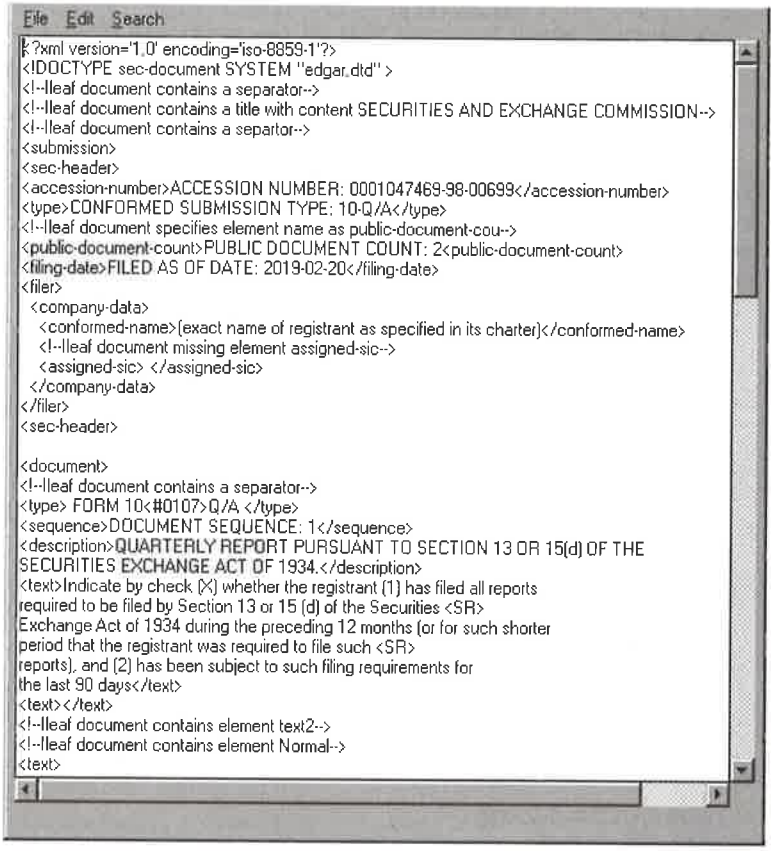


Figure 11-3 Standard plain-text XML document instance.

An XML visual editing and publishing tool could make a difference in a government filing application. Let's see how.

11.2 | An EDGAR Submission with XML

It is the beginning of your company's new fiscal quarter. As are all public companies in the U.S., your company is required to make an EDGAR submission to the SEC. You are responsible for generating the EDGAR sub-

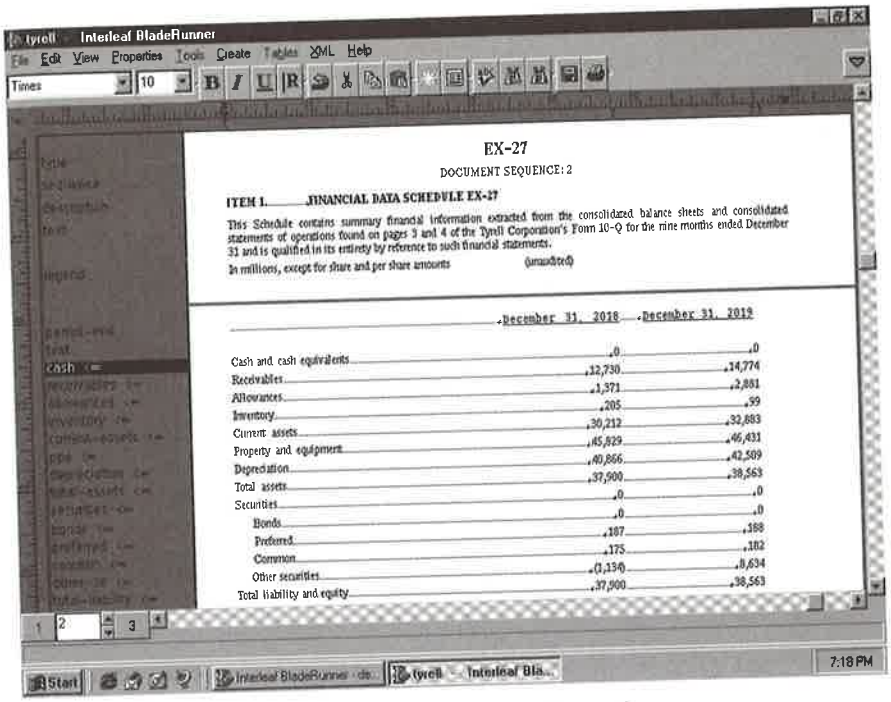


Figure 11-4 WYSIWYG XML document creation.

mission from information provided to you by the legal and financial departments.

Also, you have been asked to generate an HTML version of the EDGAR submission for your company's Web Site. However, some content from the formal SEC submission should not be include in the HTML version. The publishing challenge you face now is how do you quickly and easily generate multiple versions of the EDGAR submission with full or partial content?

11.2.1 *Reviewing the EDGAR DTD*

The SEC published the EDGAR DTD for public use. The DTD is available from the SEC Web Site. The DTD defines the required contents for an EDGAR submission and how that content must be organized. So, you obtain a copy of the DTD file. What do you do with it? How do you easily read and understand it?

In Figure 11-1 we saw what the DTD looks like in its native form. To many, the content is not user-friendly! Someone who knows how to read a DTD can use it to gain an understanding of the document structure requirements. However, in your position at your company, you are neither a programmer nor a DTD expert.

An alternative, graphical view of the DTD was shown in Figure 11-2, as presented by a visual modeling tool. With this view, the relationship of one section of an EDGAR submission to another is obvious. The optional sections are identified with question marks. With this information you are more easily able to formulate a submission that will conform to the SEC's regulatory requirements.

11.2.2 *Creating an instance of the DTD*

With an understanding of the content and structure of an EDGAR submission, you begin constructing your company's submission. Interleaf BladeRunner simplifies the task of creating an initial submission containing the minimal required sections. With a copy of the EDGAR DTD on your desktop, you can create your initial submission document (i.e. an EDGAR DTD instance) with a single click of the mouse.

Now you can begin entering the content that you obtain from the Legal and Finance departments. As you author the submission document, how can you be assured that it will conform to the SEC regulations?

11.2.3 *Checking your EDGAR instance for conformance*

As you continue to add to the EDGAR document, it is very important that the organization of your information adheres to the structure rules as defined in the DTD. Interleaf *BladeRunner* provides two modes of operation:

- "Normal mode" allows you to enter text freely into your submission document. This mode will be useful after you have become familiar with the requirements of the DTD;
- "Conformance checking" mode prevents you from creating a document that does not conform to the rules as defined by the DTD.

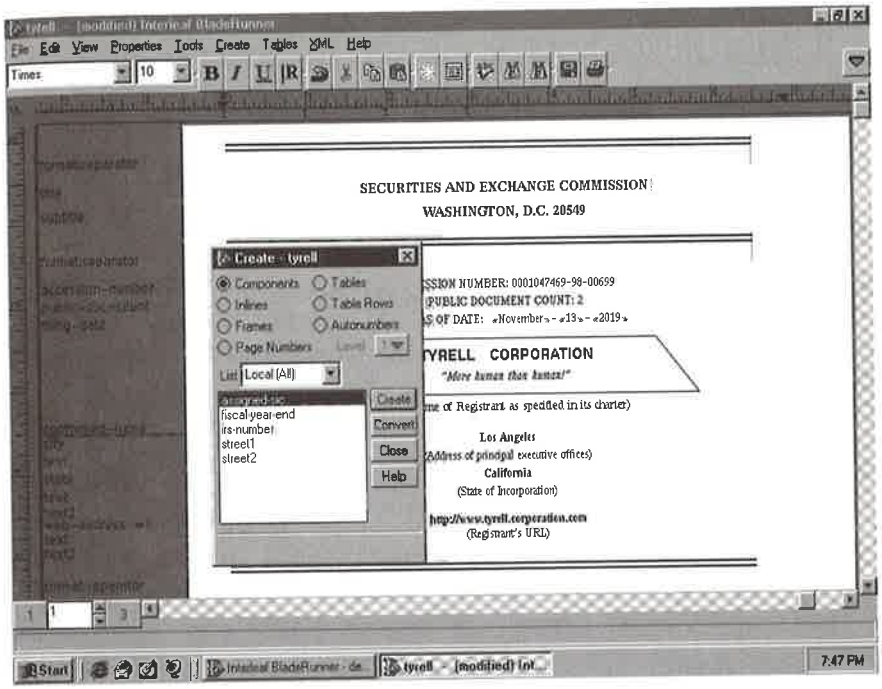


Figure 11-5 Enforcing the DTD in conformance checking mode.

As you become more experienced in creating an EDGAR document, you may find yourself switching between these two modes. Therefore, after you have created the submission document, it is important to be able to *validate* it for conformance.

As shown in Figure 11-5, *BladeRunner* provides a non-conformance error report that identifies various types of structure errors you may have created while using the normal mode of operation.

11.2.4 *Repairing non-conforming elements*

If your submission document contains structural errors, you obviously must fix those before submitting it to the SEC. *BladeRunner* includes "repair tools" for repairing structure errors. Figure 11-6

For example, one of the elements of an EDGAR submission document is "submission date". Since you are new to the EDGAR submission document type, you instinctively enter the date as follows:

```
April 5, 1998
```

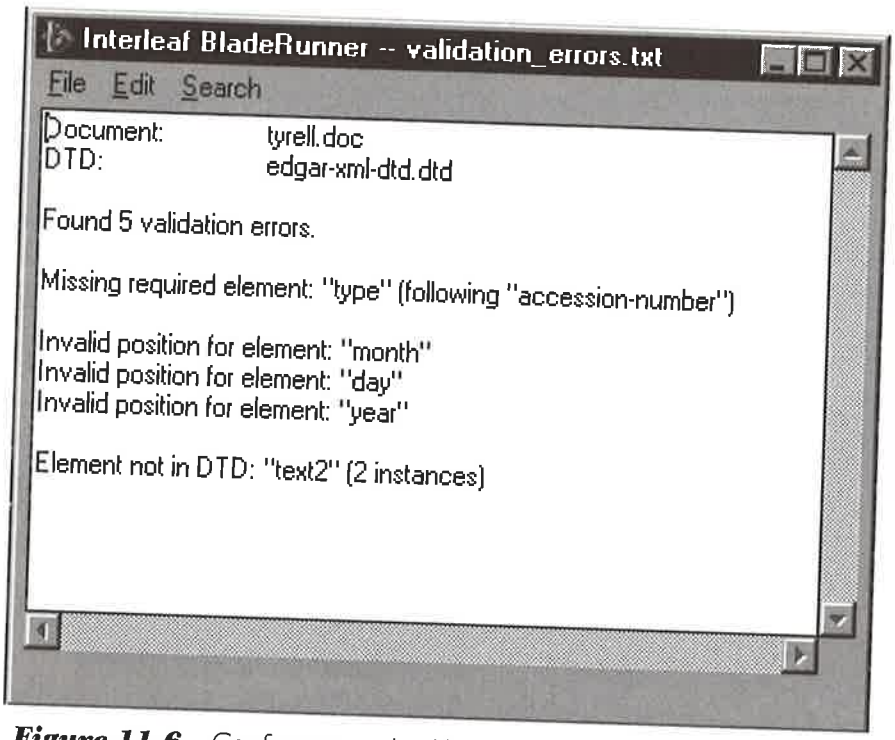


Figure 11-6 Conformance checking error report.

Figure 11-6 shows that this presentation of the submission date is "non-conforming" according to the EDGAR DTD. The submission is correctly presented as follows:

```
1998 April 5
```

Another example is the accession number, which is a required piece of information for the EDGAR document. There is a structure flow rule in the EDGAR DTD which requires that the accession number paragraph in the document must be followed by the type of document. However, being unfamiliar with the document structure, when you created the document you forgot to enter the "type" paragraph following the "accession number" paragraph.

These are examples of two non-conforming document structures which *BladeRunner* provides a utility to repair. For the first example above, you are able to use the "fix element order" utility to correctly sequence a non-conforming organization of information. For the second example, you are able

to use the "insert missing element" utility to insert some or all missing elements within the EDGAR document.

11.2.5 *Generating your EDGAR submission*

Now you have a complete and conforming submission document. The next step is to publish the document in two electronic forms: in XML for submission to the SEC, and in HTML to place on your Web site for viewing by any Web browser. Recall that for the Web site you must also remove some of the document content and reorganize the information.

11.2.6 *Publishing for the SEC*

BladeRunner includes a "Publish" feature that allows you to transform documents into different representations. You can also conditionally publish a document according to user-defined rules and specifications, according to the content of the document, and/or according to rules embedded within the document's DTD.

You publish the formal submission as an XML instance and route it to Legal for review. It can be accompanied by an XSL stylesheet that specifies the formatting and presentation rules.

11.2.7 *Repurposing for your Web site*

To produce the Web site version, you first apply conditional publishing rules to the document. In this case, the conditional rules would specify that:

1. The overview section of the EDGAR document should appear in the Web version ahead of the financial data table. In the formal SEC submission, the overview section appears at the end.
2. Information that appears ahead of the financial data table in the formal SEC submission should not appear in the Web version.

©1998 THE XML HANDBOOK™

3. Information that appears following the financial data table in the formal SEC submission should not appear in the Web version.

Next you position the EDGAR document (the same document that was used to create the formal SEC submission) adjacent to the *BladeRunner* Catalog for Web delivery.

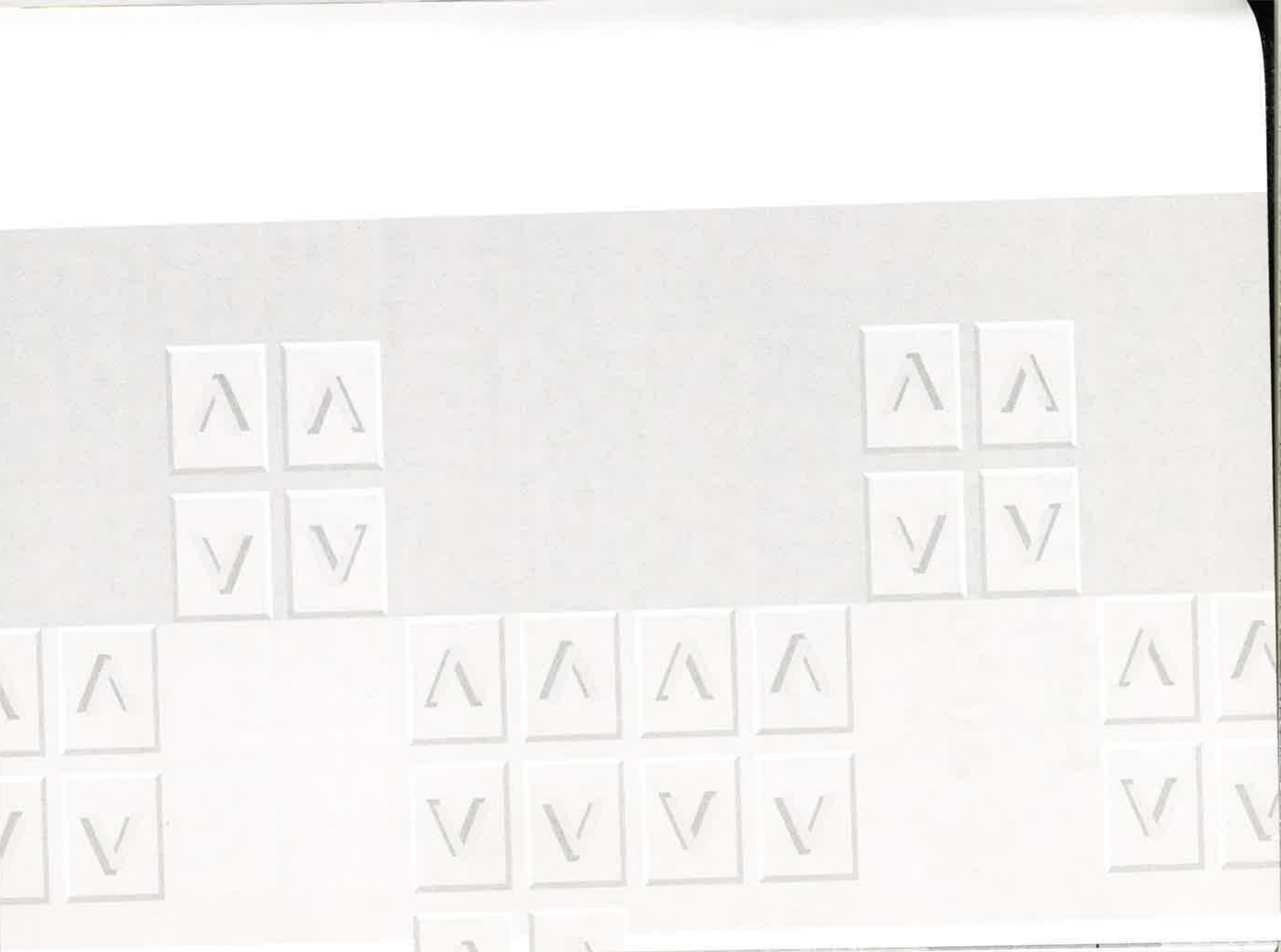
You can now use the same "Publish" feature that you used to generate the SEC submission, except this time you specify that HTML should be produced.

11.3 | Conclusion

Several business process improvements result when government filings such as EDGAR are prepared with XML and a visual document production tool like Interleaf *BladeRunner*.

1. Making the EDGAR submission in a timely fashion with assurance that the submission is complete.
2. Making company information available to the general public immediately following your formal submission to the SEC.
3. Publishing multiple presentations of the company's financial information without having to create and manage different documents for each.
4. Lowering the overhead cost of producing an EDGAR submission.

©1998 THE XML HANDBOOK™



Help Desk automation

- Content management application
- Technical support knowledge base
- Web-based extranet system

©1998 THE XML HANDBOOK™

Chapter

12

This chapter describes an XML-based technical support application that ties the Help Desk function to a dynamic repository of product information. It is sponsored by Texcel International, `http://www.texcel.com`, and was prepared by Jeremy Pollock, Derek Yoo, and Amy Krane.

Technical support is the bane of many high-tech enterprises. Yet, as products get more complicated, the Help Desk increasingly becomes the front line in the battle for customer satisfaction. But as this application demonstrates, the Help Desk might need help itself.

12.1 | The hapless Help Desk

Consider the plight of the Help Desk at a manufacturer of sophisticated, highly customized equipment that includes software-controlled electronic and mechanical components.

12.1.1 *The old way*

When our story began, the solutions provided by Support Engineers at the Help Desk varied with regard to accuracy, completeness, and applicability to the particular problem the customer faced. In addition, they were in

many different file formats, so engineers in different locations might not be able to view a solution.

The speed with which a Support Engineer could obtain a solution also varied widely. Technical fixes and workarounds were stored as whole documents on the file system, making the data in them difficult to maintain, update, and retrieve. A solution document authored by a Support Engineer could cover any number of distinct topics, but was categorized as a single file pertaining to only one topic.

Further, there was no automated way to verify the integrity of the source material and to update all instances of the same information that might occur in separate documents. Nor was it easy to share the information among Help Desk personnel, as the authoring tool was not integrated with the solution repository.

In other words, there was no way to ensure that the customer was getting the most up-to-date, correct, and personalized solution to his problem. This resulted in poor customer satisfaction and high support costs.

12.1.2 *What needed to be done?*

With the advent of ever faster and more comprehensive communications – and the Web as a strategic delivery platform – customers have come to expect fast, highly specific information. The success of the company depended on satisfied customers and functioning equipment.

To meet that need, the company realized, the Help Desk had to be linked into a knowledge repository containing the installation, maintenance, and reference documentation for the products, as well as the custom solutions developed by the Support Engineers in the course of assisting Field Engineers and customers.

Ideally, solutions should be generated that are accurate and personalized for each customer. Therefore, in this application, the Support Engineers must be provided with a tool for authoring custom solution documents, as well as the means to search for and retrieve relevant information from the knowledge base. Moreover, it is desirable to maximize the Help Desk investment by provided a controlled means to add these solutions to the knowledge repository for use by others.

Additional requirements were the ability to share and reuse information gathered in the field into the knowledge repository, providing accessible, usable feedback for the product documentation cycle. At the same time, the

organization wanted to gain portable, modular information that could be shared with its business partners and OEMs.

To show how an organization could accomplish these goals, an XML-based *Help Desk Solution System* prototype was developed. The prototype uses *Texcel Information Manager*, a content management system that is tuned to manage XML data content, element structure, metadata, and links. The product combines a dynamic document repository built on object database technology with applications for collaborative authoring, such as workflow and electronic review and commenting. Customized Java applets and integration with an existing call-tracking system comprise the balance of the prototype.

12.1.3 *Helping the Help Desk*

Consider a scenario with the prototype system in use.

In this scenario a field engineer experiences a problem with a piece of equipment installed at a customer site. He visits the manufacturer's technical support Web site, and attempts to solve his problem by searching for published technical solution information. No relevant published information is found, so the call is automatically routed to a Support Engineer at corporate headquarters who is using the Help Desk Solution System.

The Support Engineer calls up customer information from the customer tracking system, which is integrated with the Solution System, to determine the appropriate model number and other customer usage characteristics. Then he searches for applicable technical information in the dynamic knowledge repository, which consists of both published and in-process technical and maintenance documents, technical notes, training materials, and workarounds and customer solution documents submitted by other support personnel.

Although useful, the information already available and published does not specifically address the customer's problem. So the Support Engineer generates a new, custom XML solution document that combines information gained on the call, customer data, and information in the repository.

The new solution is checked into the repository, and automated system-supplied metadata is generated. Once in the repository, the solution document is available to other Support Engineers, although still in draft form.

The document is then automatically routed for review and approval via an integrated workflow system, part of the standard *Texcel Information*

©1998 THE XML HANDBOOK™

Manager product. Subject matter experts and technical editors comment on the document, using the browser-based review and commenting tool component of the product.

In addition, references to source material indicated by the Support Engineer are turned into cross-repository links to maintenance information. The new XML solution document is now available in final form for search and retrieval by other Support Engineers.

12.2 | How the *Solution System* works

12.2.1 *Information flow*

Support Engineers interact with the *Solution System* through a Java applet run from a standard Web browser. The back-end knowledge repository built using *Texcel Information Manager* contains maintenance and reference information marked up in XML.

The repository also contains other material contributed by subject matter experts in popular file formats such as *Word* and *PowerPoint*, and various graphics formats. XML metadata is associated with all objects managed in the repository. A customer support call-tracking system, running on an SQL database, also supplies data to the *Solution System*.

Support Engineers create custom, personalized solution documents using an embedded XML editing tool. The solution documents incorporate the correct product model numbers and other details from the customer records in the SQL call tracking system, which is automatically integrated into fields within the solution editor.

Once authored and approved, the solutions are categorized and stored in the repository in a way that facilitates economical querying. Each element within a solution document instance is decomposed into a unique object in the repository. A solution document instance or any element contained therein can be retrieved on demand.

©1998 THE XML HANDBOOK™

12.2.2 *Architecture*

The XML *Help Desk Solution System* consists of several client, server, and database components, as shown in Figure 12-1. The clients are a Java applet for solution research and authoring, and the *Texcel Work Queue* (an automatically generated to-do list).

The Java applet provides a GUI interface for querying the repository, and a query results display, as well as an integrated solution editor, a help Function, and a research and discovery function. The *Work Queue* delivers tasks to the user according to his role in the review and approval cycle.

The Java server application connects to multiple Java applet clients, interfaces with the document repository as well as the SQL database (through JDBC), and handles the workflow initiation and routing.

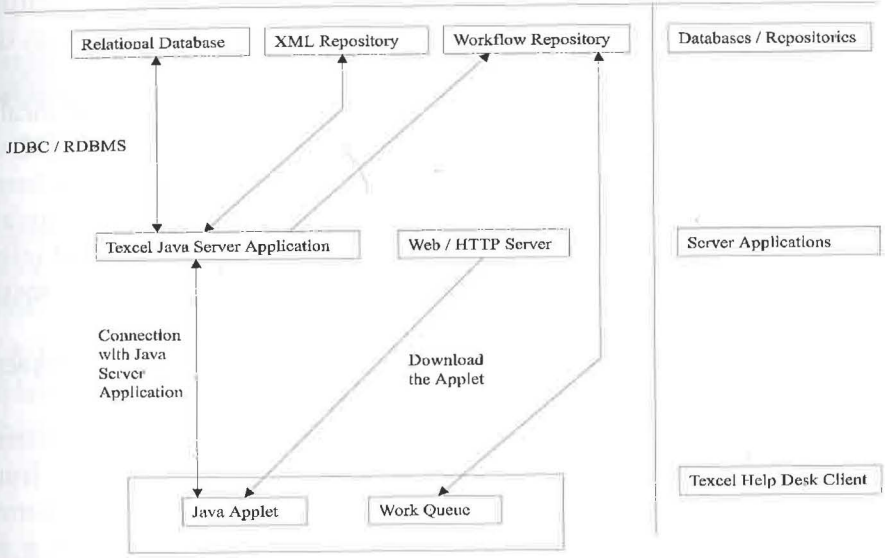


Figure 12-1 The XML *Help Desk Solution System.*

12.3 | Using the *Help Desk Solution System*

Now let's step through the application scenario and discuss the tools and technology – including the role of XML – in use at each step. We start at the Support Engineer's desk, where he is researching a solution to a problem reported by a Field Engineer on behalf of a customer.

12.3.1 *Make the query*

The Support Engineer queries the repository for solution documents that match the problem reported by the Field Engineer (Figure 12-2).

The *Help Desk Solution System* Java applet is running in a standard Web browser.

SQL queries are run against the customer call tracking system, incorporating the appropriate product name and type into the correct fields in the applet.

The search process uses the Texcel query language, which is specifically designed and optimized to query XML (and full SGML) data. The search can find information contained in any element or piece of metadata, based on any combination of element types, metadata, and data content. Searches can be run on material that is not yet released, such as solutions and reference material that is in draft form; this status information is stored as metadata.

The search tool GUI shields the user from the complexity of the query language, while generating well-formed queries and usable results.

The results of the search are presented in a tabular format in the *Solution System* applet. When an item is selected, it is converted on the fly from XML to HTML, using standard functionality of the *Texcel Information Manager* Web Application module.

12.3.2 *Research product information*

The Support Engineer researches background information in the product information repository using the *Texcel Information Manager* Explorer interface (Figure 12-3).

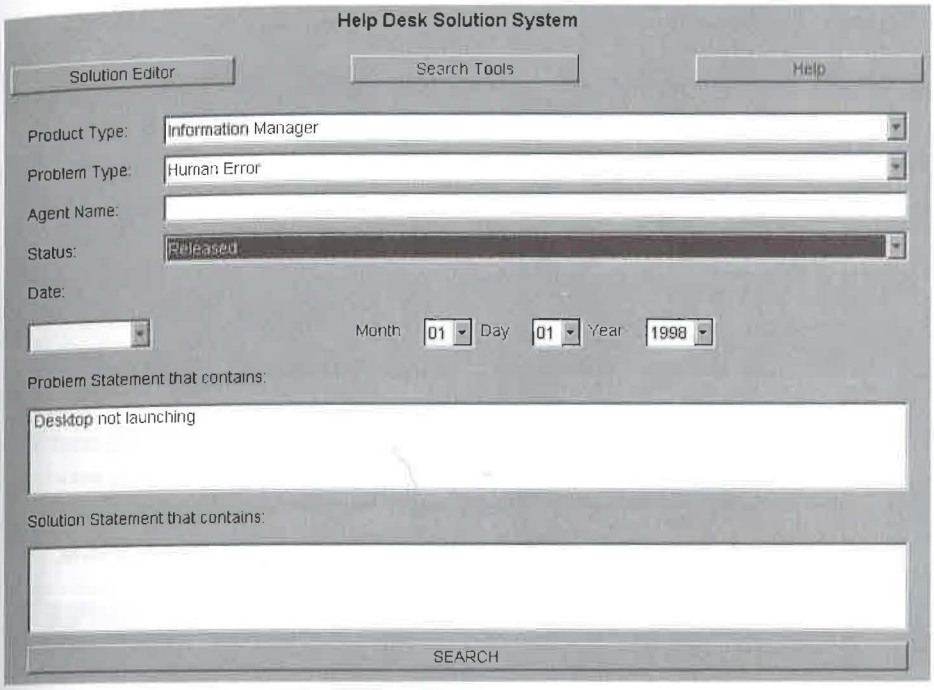


Figure 12-2 Search panel of client applet

Use of XML provides both tagged elements and metadata. The element structure can be seen in the repository browser and used for various functions. For example, a user can select a single element for updating.

A logical structure has been created for the repository, much like the folders and subfolders used in file system organization, making navigation straightforward.

12.3.3 *Write a solution*

The Support Engineer develops a new solution to the specific problem, based on interaction with the Field Engineer as well as research into the knowledge repository (Figure 12-4).

An XML document type is used that has been designed for the *Solution System* application.

©1998 THE XML HANDBOOK™

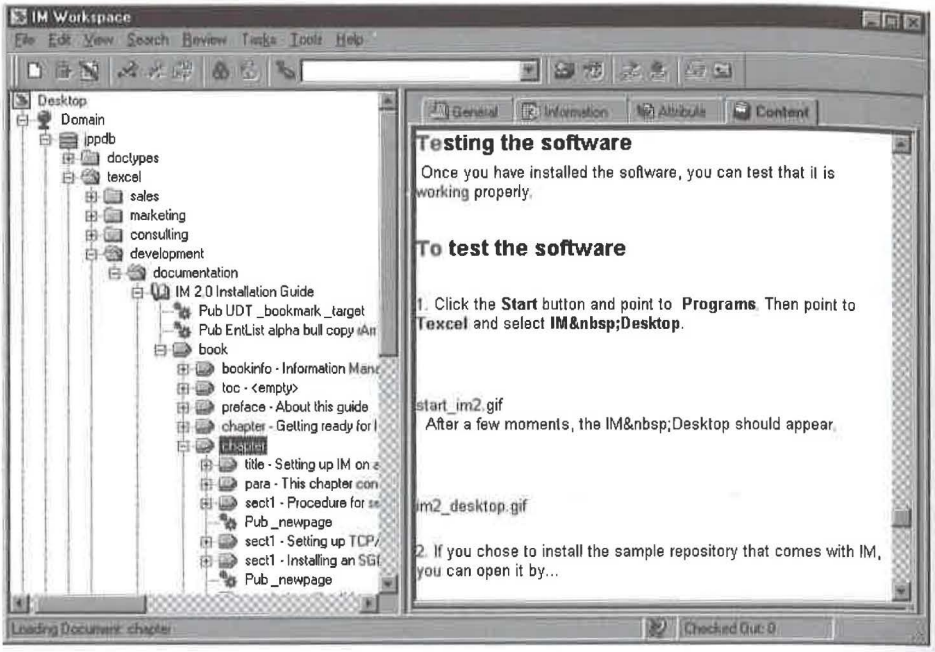


Figure 12-3 Browsing the repository for research and discovery.

Tags are not visible to the end user. Instead, the Support Engineer is able to use standard word processing functions, such as creating bulleted lists and specifying fonts.

The new document automatically includes the customer ID number and the part number of the machine that was malfunctioning, which has been extracted from the SQL database through a JDBC interface to the relational database. This information is inserted into the document and tagged with XML, as shown in Example 12-1.

12.3.4 *Update the repository*

The document is completed and submitted to the repository.

The solution document is composed and an XML parser runs to check the solution document for well-formedness.

The user selects the priority level of the document, which is used to determine which review process is used.

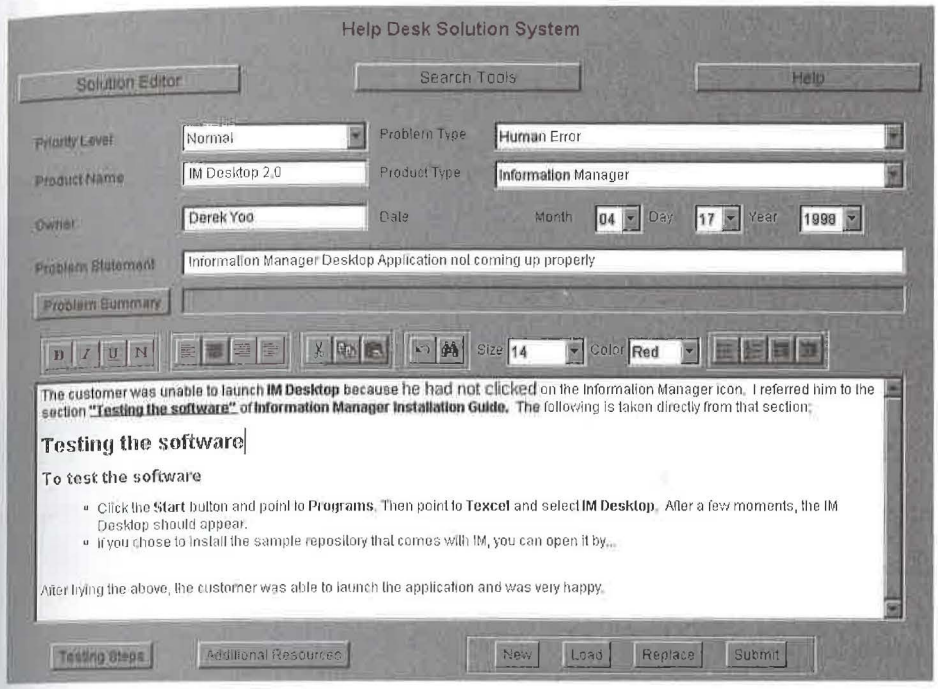


Figure 12-4 XML Solution Editor running within a Java client.

The solution document is now available to all users although still in draft or unapproved form.

12.3.5 *Route for approval*

Because the Support Engineer has authored new information, the solution document is automatically routed for approval. The routing is managed through the work queue shown in Figure 12-5.

The solution document is submitted to the Java server application, which communicates with the Texcel workflow repository.

A "case" or process is selected and the document is automatically routed to the correct people, such as the technical editors and engineers, for review and approval.

Example 12-1. Solution document marked up in XML.

```
<?XML version="1.0"?>
<solution id="solution-1000">
<solution-info>
<owner>Derek Yoo</owner>
<date>Sat Jan 31 22:30:50  1998</date>
</solution-info>
<product-grp>
<product-type>SST</product-type>
<product-name>Self Service Terminal 100A</product-name>
</product-grp>
<problem-grp>
<problem-statement> Terminal 100A will not recover from a power
failure. Screen remains blank after power is restored.
</problem-statement>
</problem-grp>
<solution-grp>
<solution-statement>
<para>The solution is to simply turn the main power switch for the
terminal off and then on again.</para>
</solution-statement>
<testing-steps>
<step></step>
</testing-steps>
</solution-grp>
</solution>
```

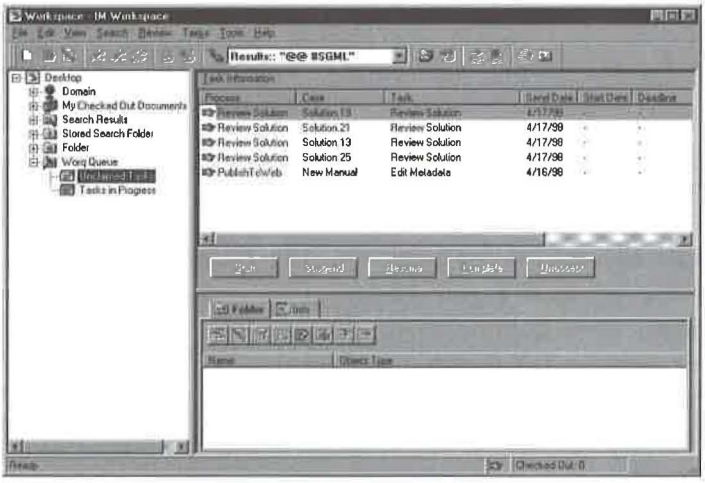


Figure 12-5 Work queue component of *Solution System*.

12.3.6 *Check in document to knowledge base*

The solution document is checked into the knowledge base and shredded into XML objects.

The check-in process parses the document and labels each element with a unique repository identifier, which can be used for retrieval and linking purposes. In addition, individual elements can be called out of the repository and reused in new solution documents as either copies of, or links to, the original information. These capabilities are part of Texcel Information Manager's support for XML.

Analysis *At first glance, this application appears to have many of the characteristics of the "large tech manual" publishing applications, perhaps because tech manuals are part of the information base. But other aspects of the application are more like classic database systems, with an emphasis on capturing bits of data, "cleaning" them, combining them with other data, and presenting them together in reports.*

XML, of course, works both ends of this street quite well, and this app demonstrates that the ends are really connected; that XML document processing and XML data processing are only different in degree, not in kind.

©1998 THE XML HANDBOOK™

Extended linking

- Extended linking defined
- XLink applications
- XPointers
- Strong link typing

©1998 THE XML HANDBOOK™

Chapter

13

Extended linking and strong link typing will let the Web traverse to locations where it has never been. Those concepts are explained simply and clearly in this chapter sponsored by ISOGEN International, http://www.isogen.com. It was prepared by Steven R. Newcomb of TechnoTeacher, Inc., http://www.techno.com, co-editor of the HyTime International Standard (ISO/IEC 10744).

uture generations of Web browsers and editors will reduce the effort required to keep our personal affairs organized and our corporate memories up to the minute. The productivity of many kinds of work will be enhanced, and in many ways. It's all going to happen basically because of two simple enhancements to the Web paradigm.

The W3C's draft XLink "extended link" facility proposes to give all of us the ability to annotate documents, and to share those annotations with others, even when we cannot alter the documents we are annotating. In other words, we won't have to change a document in order to supply it with our own annotations – annotations that a browser can make appear as though they were written right into the annotated document.

13.1 | The Shop notes application

As an example, consider a technician's set of online maintenance manuals. These are electronic books that the technician is not (and should not be) authorized to change. With the Web's existing HTML hyperlinks, the tech-

nician cannot write a note in a manual that can take future readers of that manual, including himself, to his annotations. Nor can the technician's annotations be displayed in their proper context – the parts of the manual that they are about.

13.1.1 *What is extended linking?*

By using *extended linking*, when the technician makes an annotation, he does so purely by authoring his own document; no change is made to the read-only manual document that he is annotating.

The big difference between "extended" linking and present-day HTML linking is this. With an HTML (or "simple") link, traversal can only begin at the place where the link is; traversal cannot begin at the other end. With an "extended" link, however, you can click on any of the link's anchors, and traverse to any other anchor, regardless of where the link happens to be.

Tip *Extended linking allows the starting anchor of a link to be different from the link itself. Instead of HTML's "A" tagged element that is linked to one other element, you can have (say) an "L" tag that links two or more other elements to one another.*

A simple link (top of Figure 13-1) is always embedded ("inline") in (for example) the InstallLog text from which it provides traversal; the link cannot be traversed by starting at the target anchor (for example, the Installation procedure document).

An extended link (bottom of Figure 13-1) can appear in a separate document, and provide traversal between the corresponding parts of two other documents: for example, the technician's shop notes document ("TechLog") and the read-only installation manual. Because the location of this particular link is not the same as any of its anchors, it is said to be "out-of-line" (not embedded).

In our example, an annotation takes the form of just such an extended link element.

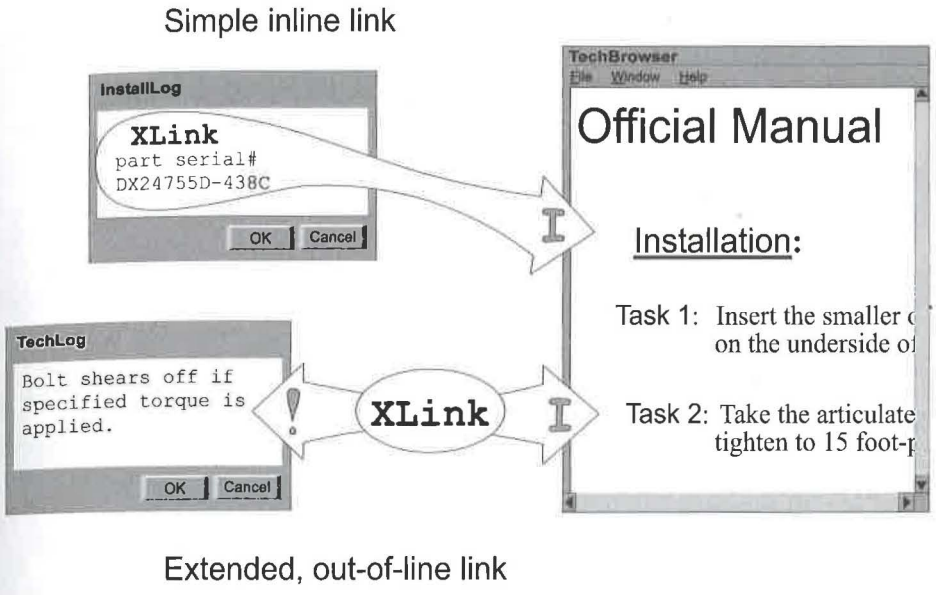


Figure 13-1 Simple vs. extended linking.

13.1.2 *Displaying extended links*

One way to realize the benefits of extended links is to display an icon at each anchor that indicates something about the other anchor. (The mechanism that supports this is discussed in greater detail under "Strong link typing", below.)

For example, as shown in Figure 13-2, a reader of the installation manual on the right will know that, if he clicks on the exclamation point displayed near Task 2, he will see a shop note about that task. If he clicks on the pound sign, he will be shown the serial number of a part that was installed according to the procedure, recorded in an "InstallLog" document.

Similarly, a reader of the annotation in the shop notes document ("TechLog") will know that clicking on the "I" icon will bring him to the installation instruction that the annotation discusses.

©1998 THE XML HANDBOOK™

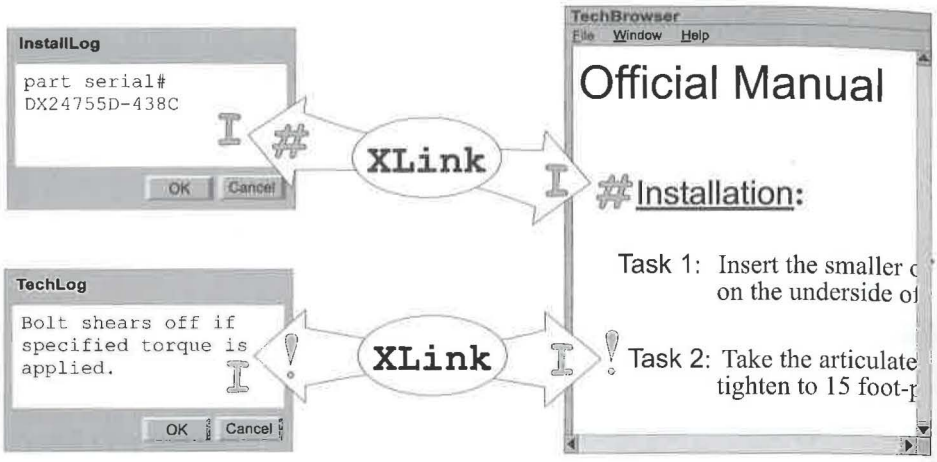


Figure 13-2 The exclamation point icon near Task 2 indicates that a shop note is available.

13.1.3 *Notes survive to new versions of manuals*

The technician's annotations – his "shop notes" – accumulate over time, and they represent a valuable asset that must be maintained. If the technician were to write shop notes inside each manual, when a new version of a manual is received it would be a chore to copy annotations from the old manual to the new manual.

With extended linking, however, the annotations are not in the old version; they are in a separate document. Therefore, the shop notes don't disappear when an annotated manual is replaced by a newer version.

That is because each link is equipped with "pointers" – pieces of information that can tell a browser where (for example) clickable icons should be rendered that indicate the availability of an annotation. Each such "XPointer" (as it is called) can point at anything in any XML document.

In our technician's shop, when a manual is replaced by a new version, the XPointers keep on working, even with the new manual, so the new manual is instantly and automatically equipped with the old manual's annotations.

In most cases, the XPointers don't have to be changed, because they continue to point at the right things, even in the new manual. If, because of differences between the old and new versions of the manual, some XPoint-

ers in the shop notes don't still point at the right things (or perhaps have nothing to point at any more), certain techniques can be used to detect each such situation. By dealing with these problem spots, the maintainers of the shop notes can minimize their efforts.

Moreover, XPointers and extended links enhance the potential for achieving high levels of quality and consistency, even when there are voluminous shop notes that annotate many manuals.

13.1.4 *Vendors can use the notes*

Some shop notes may also have value to the vendors of the manuals they annotate; they may beneficially influence subsequent versions of the manual. An editor of the manual can load (i.e., make his browser aware of) all the shop notes of many repair shops; this has the effect of populating the manual with icons representing the annotations of all the shops. The most common trouble spots in the manual will be made obvious by the crowds of annotation icons that they appear to have accumulated (Figure 13-3).

The fact that the shop notes take the form of interchangeable XML documents that use standardized extended links makes the task of sharing internal shop notes with manual vendors as easy as sending them any other kind of file. There is no need to extract them from some other resource, or to format them in such a way that they can be understood by their recipients. They are ready to work just as they are, in the tradition of SGML, HyTime, HTML, and now XML.

13.2 | Other applications of extended linking

The above "shop notes" example is just a sample of the kinds of enhancements that extended linking will bring to our interactions with information resources. Some of the broader implications are a bit more startling.

©1998 THE XML HANDBOOK™

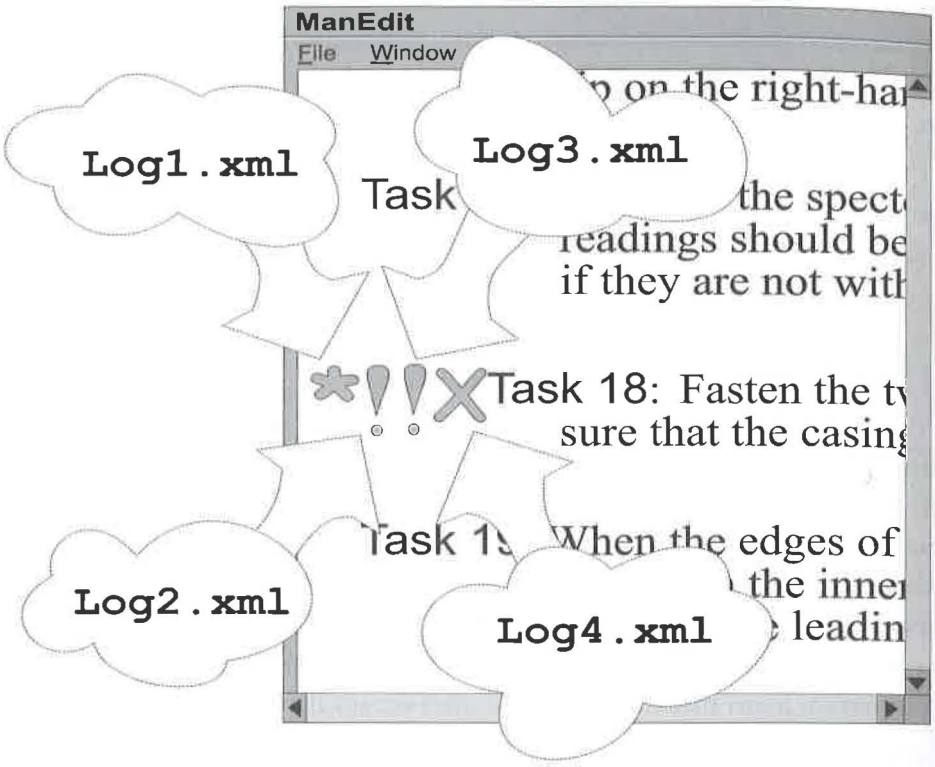


Figure 13-3 Task 18 evidently prompted three kinds of annotations in four different shop logs.

13.2.1 *Public resource communities of interest*

For example, many web sites today contain HTML links to public resources. One is the U.S. Government's online service for translating any U.S. postal code into its corresponding Congressional district and the name of its current incumbent Representative (`http://www.house.gov/zip/ZIP2Rep.html`).

However, if those HTML links were to become XLink extended links, an XLink-enabled browser could render this U.S. government Web page in such a way as to add to it a catalog of the activists and lobbying organizations who refer readers of their websites to this particular U.S. government

resource. The "marketplace of ideas" represented by the aggregate of such organizations is thus revealed in a new and interesting way.

13.2.2 *Guidance documents*

Another startling possibility is the association of browser-controlling metadata with any and all Web resources.

In this scenario, a document of annotations (or a set of such documents) can be a user's companion during excursions on the Web. These annotations might make suggestions to users as to where to find more recent material, or they might even take control of the browser's link traversal ability in order to protect children from disturbing material.

While the latter XLink-enabled possibility may sound inimical to the freedom of speech, in fact it enhances liberty. It provides a new public medium for free speech: documents that censor the Web and/or otherwise provide guidance to Web travelers in the form of annotations that appear only in their designated contexts.

Of course, no adult is required to use any such guidance document, just as no one is required to read any particular book, but it's easy to predict that many will pay for the privilege of using many kinds of such "guidance documents."

More importantly, everyone will have the tools to write such guidance documents, so the technical ability to provide guidance (and, yes, even to provide censorship services) will be widely distributed, rather than being dangerously concentrated in a few generalized rating services. The creation and maintenance of guidance documents may well become a thriving cottage industry. Anyone can be a critic.

In the case of electronic commerce, it's easy to imagine that vendors will attempt to provide guidance documents designed to annotate the online sales catalogs of their competitors. In response, some providers of online sales catalogs will take steps to render the pointers in these kinds of guidance documents invalid and unmaintainable.

Regardless of all this, the overall impact on electronic commerce will certainly be positive; increasing the meaningful interconnectedness of the Web will help more people find exactly what they're looking for.

And it may turn out to be a mistake, in many cases, for catalog owners to attempt to render the pointers used to annotate their catalogs invalid, because similar pointers could be used, for example, by impartial consumer

©1998 THE XML HANDBOOK™

testing organizations to attach "best buy" recommendations to certain products. The guidance documents of consumer testing organizations will probably be quite popular, and well worth the cost of using them.

13.2.3 *Computer-augmented memory*

Extended linking has the potential to make radical improvements in our ability to keep track of what we are doing. Someday, we can expect to automatically annotate each piece of information we work with in such a way that, in effect, it refers future readers to the work we did with respect to it.

In other words, practically everything we do can be usefully seen as an annotation of one or more other pieces of work. If everything we do is, in some sense, an annotation of one or more other things, everything we do can all be found far more easily, starting from any piece of work anywhere in the "chain" (or, more likely, "tree" or "graph") of relevant information.

This is because extended linking allows all links to be bidirectional. (Or, rather, "n-directional", to account for extended links with more than two ends.) All of the connections among our affairs can then be tracked more or less automatically, so that each of us can enjoy a radical reduction in filing, cross-indexing, and other organizational chores, and with vastly increased ability to find what we're looking for quickly and easily.

Obviously, this same idea is even more significant in the realm of corporate memory. Even with today's behemoth enterprise integration technologies, it's still too hard to figure out what has happened, who is doing what, how various plans and projects are going to integrate, and where the relevant paperwork can be found.

Going a step further, there is an ISO standardization activity (ISO CD 13250) seeking ways to exploit extended linking in such a way as to create living, easily explored and maintained "maps" of all of the information resources available to an organization (see "Topic Navigation Maps", `http://www.hightext.com/tnm`). This goal sounds almost insanely ambitious, but extended linking, in combination with strong link typing (see below), could make it practical and achievable.

13.2.4 *Intellectual property management*

The advent of extended linking also offers interesting new possibilities for the management and exploitation of intellectual property.

For example, metadata regarding the licensing policies of owners of Web resources could be associated with those resources by means of extended links. Such metadata could be changed when the resources are sold or licensed, without requiring any changes to the assets themselves.

This method greatly reduces the likelihood of inadvertent damage to the assets, and greatly increases the ease with which ownership and/or management policies can change. There is already an official, internationally-ratified ISO standard for using extended linking for exactly this purpose (see `http://www.ornl.gov/sgml/wg8/document/n1920/html/clause-6.7.html#clause-6.7.3`).

Such activity policies, and the means by which they are associated with online assets, could well become a source of private law that will strongly influence the development of intelligent agents (see `http://www.hytime.org/papers/higgins1.html`).

13.3 | Strong link typing

With the XLink extended link facility, there is no limit to the number of links that can be traversed from a single point in a single document. Many different documents can contain links to the very same anchor, with the result that, theoretically, at least, an unlimited number of traversals are possible, starting from a single point. In addition, there are no limits on the kinds of annotations that can be made, nor on the purposes to which such annotations may be put.

Therefore, it makes sense to provide some easy way to sort the annotations (i.e., the links) into categories. For example, some kinds of annotations will be made in order to provide "metadata" about the document, and these will often take effect in some way other than by rendering an icon on the display screen. Some kinds of annotations are interesting only for specialized purposes.

©1998 THE XML HANDBOOK™

13.3.1 *Hiding the installation log*

Going back to our earlier example, the technician can create an annotation that indicates the serial number of a new part that he installed in accordance with a particular maintenance procedure. The fact that such an annotation is available would be of interest only to someone who was auditing the installation of parts; it probably wouldn't appear even to the technician, despite the fact that it was he who created the annotation.

The technician's installation log annotation can be hidden from most people because it is "strongly typed": it has been clearly and unambiguously labeled as to its intended meaning and purpose, so all browsers can see what kind of link it is. In effect, the link says, "I am a Part-Installation-Log-Entry." People who aren't interested in part installation records can arrange for their browsers to hide them.

13.3.2 *Why do we need strong link typing?*

People may still choose to be made aware of other kinds of annotations made by our technician. For example, other technicians may wish to read our technician's accounts of any special situations that he has experienced when attempting to follow a particular instruction, or about successful and unsuccessful experiments with substitute parts.

The notion of "strong link typing" is virtually absent from HTML links. Basically, in HTML, the browser software knows where the user can go, but not why the author of the document being browsed thought the user might like to go there. The human reader can usually divine something from the context about the material that will be shown if the "anchor" hyperlink is traversed, but the browser itself is basically unable to help the user decide whether to click or not to click, so it can't hide any available traversals.

To be able to hide the availability of unwanted kinds of links can save a lot of time and effort. So the draft W3C XLink recommendation also provides for the addition of strong typing features, not only to extended links, but also to the "simple" links that closely resemble the familiar HTML "anchor" (<a>) element. Thus, browsers can start supporting strong link typing promptly, even before they can handle extended linking.

13.3.3 *Anchor role identification*

The notion of strong link typing includes the notion of "anchor role" designation.

For example, the simple link at the top of Figure 13-1 characterizes its target anchor as an installation instruction; in the diagram, this is indicated by the "I" icon in the arrowhead. Similarly, the extended link at the bottom of Figure 13-1 characterizes one of its anchors as a shop note (the exclamation point) and the other anchor as an installation instruction (another "I" arrowhead).

Thus, a link can do more than just identify itself by saying, for example, "I am a Part Installation Log Entry." It can also specify which of its anchors fulfill which roles in the relationship it expresses.

For example, our Part Installation Log Entry link can say, in effect, "I signify that part [pointer to entry in parts catalog or inventory record] was installed in [pointer to information that identifies the unit being maintained] in accordance with maintenance directive [XPointer to instruction in manual]."

In other words, the log entry link is a three-ended link whose anchor roles might be named "replacement-part" (indicated with a "#" icon), "maintained-unit" ("@" icon), and "maintenance-directive" ("I" icon) (Figure 13-4).

The fact that an anchor plays some specific role in a relationship often determines whether the relationship is interesting or even relevant in a given application context.

13.4 | Conclusion

It is easy to see that the impact of extended linking will be significant, and that technical workers and electronic commerce will be early beneficiaries. Extended linking will enhance the helpfulness and usefulness of the Web environment. The burden of many kinds of paperwork will be very substantially mitigated.

On the horizon, there appears to be serious potential for significant improvements in the availability of all kinds of knowledge, due to the possibility of creating and interchanging Topic Navigation Maps. Intellectual

©1998 THE XML HANDBOOK™

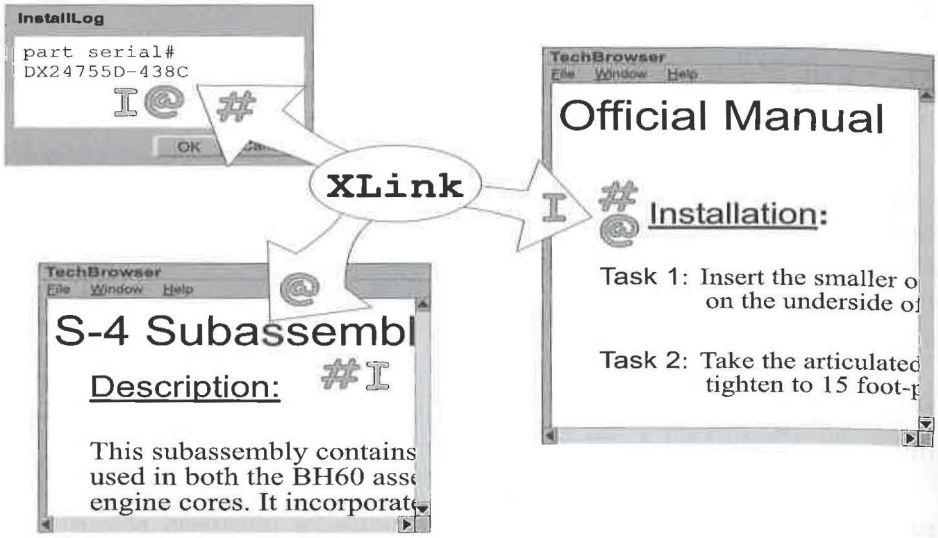


Figure 13-4 Link with two traversal possibilities at each anchor, distinguishable because of anchor role identification.

property management, and the Web-based utilization of intellectual property, will become easier and more orderly.

All of these benefits, and probably many more, emanate from two very simple enhancements of the Web paradigm in the draft XLink and XPointers recommendations of the World Wide Web Consortium:

- Allowing the starting anchor of a link to be different from the link itself; and
- Strong link typing, in which links plainly exhibit the kind of relationship they represent, and the roles their anchors play in that relationship.

Tip *For more on XLink and XPointers, see Chapter 34, "Extensible Linking Language (XLink)", on page 498. The text of the XLink and XPointer drafts are on the CD-ROM.*