CHAPTER 1

A Brief History of Electronic Data Interchange

Unlike most of its siblings from the Microsoft .NET Enterprise Server family, Microsoft BizTalk Server 2000 (BTS) is not a product that you can install, configure, and run within a few hours, and it is not as instantly usable for end-user scenarios as Microsoft Exchange Server 2000 for electronic mail and collaboration tasks, or SharePoint Portal Server 2000 for corporate intranet portals and document sharing. Instead, Microsoft BizTalk Server 2000 is one of the first Microsoft products specifically designed for the needs of enterprise customers, who want to leverage the power and benefits of direct business process and systems integration across corporate boundaries. As such, BizTalk is much more of a development and integration toolkit than a "traditional" server product that supplies end-user applications like Microsoft Internet Explorer, Microsoft Outlook, or your internal line-of-business applications with data or documents. From the corporate end-user's perspective, BizTalk is a "stealth" product that silently and powerfully forwards digital documents between applications and business partners, but will go largely unnoticed by many users.

The fact that BizTalk is a product that acts much in the background and has no flashy "end-user ready" interfaces and does not do live streaming media should not distract you from a very important fact: Microsoft BizTalk Server 2000 is one of the centerpieces of Microsoft's .NET strategy and Microsoft's central hub for digital business document exchange.

Because BizTalk is not a "plug-and-play" product, the first part of this book is entirely dedicated to providing you with the necessary background and reasoning for why electronic document interchange is important for your business and how the web services paradigm that is part of Microsoft's third-generation Internet vision will change the way electronic business is conducted.

CASTRUM NOVAESIUM

We're starting our exploration of Microsoft BizTalk Server 2000 at a very unlikely place and a very unlikely time: Castrum Novaesium, A.D. 50. To maintain an iron grip on their empire, the Roman emperors and their administrative apparatus had a system that could be met in its speed and efficiency only in the late nineteenth century: the *cursus publicus*, or Roman postal service.

Castrum Novaesium was a Roman military settlement on the west shore of the Rhine River, located on the territory of the modern city of Neuss, Germany. Messages from Rome could reach the local commander in as little as eight to ten days and those from the province governors within only a day or two, traveling at speeds of up to 270 km per day. This speed enabled the Romans to quickly learn about problems in their colonies and to order redeployment of troops within a matter of days—much faster than any of their adversaries.

Another pillar of Rome's success was their bureaucracy. Organizational orders and military messages were strictly formalized and brief, expressed in a very logical and unambiguous language: Latin. The orders were transported using the reliable military postal system and were uniquely signed and authenticated.

If the Novaesium commander encountered problems with the Germanic barbarians on the other side of the Rhine river, he could efficiently ask for help and quickly receive it. This advantage was essential not only to maintain power, but also just to stay alive. At the same time, Roman citizens could maintain a most enjoyable life back home due to the efficiency of the administration and tax collection system in place even at their remotest provinces—enabled by a sophisticated communication system.

The organizational level of the Roman postal system was hardly met, well into the nineteenth century. However, the diplomatically correct formalization of letters—along with the strict distinction between address and handling information, content, and signature to guarantee secrecy and authenticity—are still important to governments today. Are you surprised that we're going to rediscover all of these features when we explore electronic data interchange (EDI), eXtensible Markup Language (XML), Simple Object Access Protocol (SOAP), and BizTalk Messaging?

But wait—reliable, quick, and secure messaging isn't everything. Orders and organizational measures that are triggered by incoming messages must be carried out in a well-defined and orderly fashion.

In times of crisis, like heavy losses of soldiers in battle or poor farm seasons in the homeland, emperors appeased the Roman citizens not only through *panem et circencis* (bread and games), but also by increasing the tax deeds of the provinces. Fulfilling such a vital, central role required the emperors to orchestrate all of the empire's resources by giving orders and setting up procedures: military camps had to be put on higher alert to avert uprisings in the conquered territories, special envoys were sometimes sent to assist and supervise the effort, and local governors had to execute the orders thoroughly with their local forces.

However, once the mandated taxation level was established, the empire had to be flexible enough to reallocate resources and switch organizational structures quickly to conquer and assimilate more territories to further enhance Rome's wealth. If the empire had a static, military structure optimized only for rapid expansion, it would have suffered the same quick collapse as all those later attempts to conquer the known world. The Romans were successful through communication and orchestration of all their resources—for more than five centuries.

BizTalk will neither enable you to conquer the known world nor guarantee you a successful business for the next few centuries; however, its Orchestration features will allow you to adjust your document-driven business processes quickly to meet changing needs. With BizTalk, you can visually create dynamic workflow schedules instead of writing and maintaining static and manually coded (and therefore expensive) programs or batch jobs. Its messaging features will let you accept and deliver messages from your outposts and allies and act rapidly when the volume or shape of the information changes, or when you need to add new corporate settlements or partners to your information flow.

ELECTRONIC DATA INTERCHANGE

Let us move on to some more recent history and enter the electric age.

The telegraph's immediate predecessors—light-signal towers—had been used in England and France since the late eighteenth century. They allowed quick relay of manually keyed signals from hilltop to hilltop. But the Morse telegraph, which started spreading quickly in Europe and North America in the 1840s, was a truly revolutionary invention. It allowed information to travel at the speed of light end-to-end and at an affordable cost.

This innovation made the telegraph one of the earliest and most important commercial applications of electricity for communications. Because commercial success spawns competition and money, only roughly 30 years after the invention of the telegraph, Phillippe Baudot invented a five-bit code that was used for transmitting character information between machines. His invention combined the typewriter and the telegraph and later became commonly known as the teletypewriter, Teletype, or Telex.

The Teletype and Telex systems grew to span the world in the early twentieth century and became ubiquitous in the 1940s, allowing businesses to communicate and exchange information at 150 Baudot, or 30 characters per second, allowing a full page of information to be transmitted in about two minutes.

While Telex allowed the transfer of typed documents between locations, it was still a paper-to-paper technology that was not really designed for exchanging raw data. The need to exchange raw data grew with the increasing use of digital computer systems in commerce. Large international corporations like retailers, transportation providers, and manufacturers started to establish direct data links between systems in the mid-1960s. They were driven by the vision of the "paperless office" and more reliable communication that eliminated human error in transferring printouts manually from one system into other systems.

Although these first steps of electronic data interchange (EDI) proved to be efficient, they were proprietary. Thus, they were costly to implement and nearly impossible to deploy across a large number of organizations. Recognizing this, the Accredited Standards Committee X12—a standards institution under the umbrella of ANSI, appointed by EDI vendors and users—moved to standardize the EDI space.

As a result, the ANSI ASC X.121 (ANSI X12) standard was established, which has been the dominant EDI document format for the past three decades. The following is a sample purchase order document as it looks when expressed in ANSI X12 ("transaction set" 850):

```
ISA*00*    *00*    *01*003897733    *12*PARTNER
ID*980923*1804*U*00200*000000002*0*T*@
GS*PO*MFUS*PARTNER ID*19980924*0937*3*X*004010
ST*850*0001
BEG*00*SA*4560006385**19980923
CUR*BY*USD
TAX*1-00-123456-6
FOB*DF***02*DDP
ITD*01*ZZ*****45*****NET 45 - Payment due 45 days from Document Date
TD5*Z****Ship via Airborne
N9*L1**SEE FOLLOWING TEXT
```

```
MSG*PLEASE CONFIRM PRICE IF NOT CORRECT.
N9*L1**SEE FOLLOWING TEXT
MSG*THANKS, BILL 281-555-5555
N1*BT**92*USA1
N1*BY*BLUEWAY SAMPLE CORPORATION*92*MFUS
PER*BD*PETE WELLDONE
N1*SE*PARTNER COMPANY NAME*92*0010001000
N1*ST*Blueway Sample Corporation*92*0000002924
N3*24500 F.D.R. Drive 290
N4*New York*NY*10004*US
PO1*00010*3300*EA*0.888*CT*BP*123456-123*EC*AM*VP*123456*123
PID*F****LOGO, 2000,MC-R6
SCH*3300*EA***002*19981101
CTT*1
SE*23*0001
GE*1*2
IEA*1*000000002
```

ANSI X12 also forms the basis for the international UN/EDIFACT (United Nations Directories for Electronic Data Interchange for Administration, Commerce, and Transport) standard, whose definition started in 1985. UN/EDIFACT has been officially phasing out ANSI X12 since 1995; however, the ANSI X12 standard remains so popular in the United States that the final normative adoption of UN/EDIFACT instead of ANSI X12 is postponed year by year by year. The fact that the United Nations is developing a technology standard clearly signals the importance of EDI in international commerce.

The following document is also a purchase order. It is expressed as a UN/EDIFACT "ORDERS" message that is compliant to the standards release S93A (directory release D93A):

```
UNB+UNOB:1+003897733:01:1234567+PARTNER ID:ZZ+000101:1050
+00000000000916++ORDERS'
UNH+1+ORDERS:S:93A:UN'
BGM+221+P1M24987E+9'
DTM+4:20000101:102'
FTX+PUR+3++GENERAL PURCHASE ORDER INSTRUCTION'
RFF+CT:123-456'
RFF+CR:1'
NAD+SE+10025392::92++SUPPLIER NAME'
CTA+SR+:STEVE'
NAD+BT+B2::92++BLUEWAY SAMPLE CORPORATION+P O BOX 901000
+NEW YORK+NY+10013000+US'
NAD+BY+MFUS::92++BLUEWAY SAMPLE CORPORATION'
CTA+PD+:JANE EGGLAND-FULTON'
NAD+ST+CM6::92++BLUEWAY SAMPLE CORPORATION+CCM6 RECEIVING DOCK:20555
```

```
SH 249+NEW YORK+NY+10034+US'
TAX+9++++++3-00105-5135-3'
CUX+2:USD:9'
PAT+1++1:1:D:45'
PAT+22++1:1:D:30'
PCD+12:2'
TDT+20++++:::AIRBORNE'
LOC+16+BLUEWAY DOCK'
TOD+2+NS+:::ORIGIN COLLECT'
LIN+000001++107315-001:BP'
PIA+1+AA:EC+123456:VP'
IMD+F+8+:::PART DESCRIPTION INFORMATION'
QTY+21:10000000:PCE'
DTM+2:20000301:102'
FTX+PUR+3++LINE ITEM COMMENTS'
PRI+CON:50'
TAX+7++++:::100'
MOA+124:100'
UNS+S'
UNT+31+1'
UNZ+1+000000000000916'
```

Before the Internet even left the educational space, value-added networks (VANs)—operated by providers such as IBM, GEIS, and AT&T—have been connecting the Global 2000 companies, reliably carrying EDI messages, optimizing collaboration, and reducing costs in international business.

In finance, EDI plays such a central role that the global financial system wouldn't function without it. The Society for Worldwide Interbank Financial Telecommunications (SWIFT) was initiated by seven major international banks in 1974 to overcome the speed, format, and security limitations of the Telex system. The society established the SWIFT messaging system in 1977, when it started operations with 230 banks from five countries.

SWIFT defines a large number of standard messages based on the ISO 7775 standard, which covers most aspects of interbank communications. (ISO 7775 is currently being phased out by the incremental ISO 15022 standard.) One of the better-known SWIFT messages is MT940, the "customer account statement" message that can be imported by most business accounting systems.

Here is a sample SWIFT message:

```
{1:F01BANKBEBBAXXX2222123456}
{2:I100BANKDEFFXXXXU3003}
{3: {113:9601}{108:abcdefgh12345678}}
{4:
:20:PAYREF- TB54302
:32A:910103BEF1000000,
```

```
:50:CUSTOMER NAME
AND ADDRESS
:59:/123-456-789
BENEFICIARY NAME
AND ADDRESS
-}
{5:{MAC:41720873} {CHK:123456789ABC}}
```

SWIFT's rock-solid, secure network carries over 3 million messages a day with an uptime consistently exceeding 99.99 percent a year. It is so reliable that SWIFT takes full responsibility for messages once they have been accepted and are transmitted over its network.

Ninety percent of U.S. Fortune 500 companies use UN/EDIFACT or X12 solutions, with similar adoption levels in Europe. Roughly only 6 percent of all other companies in the world are EDI enabled. However, over one hundred thousand businesses in the United States use an EDI solution of some sort to communicate with their customers, suppliers, or financial service providers.

THE EXTENSIBLE MARKUP LANGUAGE

While EDI was bred in the domain of global corporations, the eXtensible Markup Language (XML) is the child of the personal information technology space. Ever since Apple invented the legendary Apple II and IBM introduced the PC (with the short-sighted intent of creating a smarter mainframe terminal), the development of personal computers and their software was mainly driven by consumers and small businesses.

Most of today's software developers grew up with PCs or desktop workstations using Microsoft's DOS, Windows, or some variant of UNIX. The PC revolution concentrated on the desktop, providing personal or group-enabled network solutions for businesses, home organizations, and entertainment.

The desire for information, natural human curiosity, and the potential for entertainment were also the driving forces for the rise of the Internet. Conceived in the 1970s, the Internet became a universal network for educational institutions around the world throughout the 1980s with the emergence of Hypertext Markup Language (HTML). HTML combines a reduced text layout description variant of the Structured General Markup Language (SGML), a document description and definition language developed for the publishing and printing industry, and the simple concept of the hyperlink to provide navigable (clickable) pointers between documents. No longer an overly complex tool only for computer geeks, the Internet grew rapidly in the educational and scientific sector with the popularity of the HTML-based World Wide Web. In 1994, the Internet finally exploded like a giant supernova into the commercial space.

HTML is a simple, text-based format with just a handful of syntactical elements that allow you to define the layout of text documents. HTML was originally developed in conjunction with the Hypertext Transfer Protocol (HTTP) to allow international scientists to

efficiently access the large amounts of research data that the Swiss nuclear research lab CERN produces every day.

HTML's simplicity and platform-independence enables users to easily publish their ideas and access others' ideas without requiring them to install specialized applications—except a single, simple HTML viewer-application: a browser. With hyperlinks, all of this information can be woven together to create an informational web. What a phenomenal idea. It's so phenomenal that it created an entire new industry—driven by enthusiastic PC-agers.

Take the simplicity and the most fundamental text notation of HTML, introduce some more consistency, and then drop all of the hypertext layout elements in favor of a mechanism to allow everyone to come up with their own set of elements, and you get XML. The following is a sample XML data stream:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<CustomerData>
   <Customer ID="1234567">
      <Name>newtelligence AG</Name>
      <Contacts>
           <Contact>
                <Name>Vasters</Name>
                <FirstName>Clemens</FirstName>
                <Position>CTO</Position>
           </Contact>
           <Contact>
                 <Name>DePetrillo</Name>
                 <FirstName>Bart A.</FirstName>
                 <Position>
                     Director Intl. Business Development
                  </Position>
           </Contact>
      </Contacts>
      <MailAddress>
          <Country>DE</Country>
          <PostalCode>41352</PostalCode>
          <City>Korschenbroich</City>
          <Street>Gilleshütte 99</Street>
      </MailAddress>
    </Customer>
</CustomerData>
```

Traditional EDI formats structure their content either by using predefined positions or by using certain—often user-defined—delimiter characters. Common to all EDI data formats is that external documentation is required to understand, parse, and write any EDI document. XML is not only much simpler, but it is also very self-descriptive.

In fact, the revolution that HTML sparked for human-readable interactive text and information is synonymous with what XML is doing for data. XML is a young standard: the base specification was standardized in 1998, and a lot of the more advanced specifications advanced to the standard-equivalent recommendation level only by the end of 2000.

XML has rapidly grown beyond the simple set of text formatting rules defined in the XML 1.0 recommendation issued by the World Wide Web Consortium (W3C). Therefore, XML is attractive for virtually every aspect of computing and is growing more popular even in areas traditionally covered by EDI. Every advanced XML feature in the following list has no equivalent match in any of the popular EDI formats. The XML format is clearly superior in terms of syntax and universal applicability.

- ▼ Namespaces allow you to merge multiple sets of semantics into a single document. XML documents can carry additional user-defined data and control information at any place, without violating the semantics and integrity of a standardized specification.
- ■ Schema allows you to narrowly define the structure of an XML document in XML itself, making the XML documents and their definitions entirely self-contained. Schema also specifies a rich set of standard data types that can be user-refined with precise constraints or grouped into arbitrarily complex types.
- ■ The XML Information Set defines an abstraction of the XML data format, providing you with a standardized in-memory representation of XML documents; it enables the creation of a unified programming interface for XML documents: the XML Document Object Model (DOM).
- ■ XBase and XLink allow you to define precise locations and cross-references between any element inside or across documents. The XPath language, which builds on these standards, is a powerful language for expressing queries into XML documents.
- ■ Stylesheets provide you with a standardized mechanism to define transformation rules between different document types (schemas). They also provide a way to augment XML data with formatting hints for improved presentation to human readers.
- ▲ The Simple Object Access Protocol (SOAP) and the upcoming XML Protocol (XMLP) (which is based on the SOAP proposal) specify a standard envelope and transport bindings so that you can move XML data between systems in a standardized way.

In contrast, all common EDI standards define only the core construction rules for messages. They also have truckloads of static data dictionaries, expressed in a nonstandardized mix of tabular field definitions and prosaic explanations that define the permitted records, fields, and their relationships.

None of the XML specifications depends on a certain platform or implementation. Because simplicity is a fundamental principle of the core XML standard, all of the aforementioned specifications are designed in the same spirit. Consequently, any software

development organization can easily implement XML and all of its facets on every platform and any device, ranging from large mainframe systems to portable phones or "wearable computers."

ENTERPRISE APPLICATION INTEGRATION WORRIES

Simplicity sets XML apart from all other previous attempts to establish standardized application-to-application data interchange, recently more often referred to as enterprise application integration (EAI). Before XML became popular, the integration technologies developed in the 1990s were basically divided into four different camps, which were rapidly drifting away from each other with only incomplete and brittle bridging technologies between them.

All of these technologies provide remote access to business logic services (objects) and provide ways to publicize and find these services on networks. They also provide additional services for, amongst others, authentication, authorization, privacy, and transaction support.

The EDI space is one of the camps, without necessarily being aware of it. EDI traditionally has been a mainframe computer domain that everyone had to integrate toward and that would typically not actively integrate other systems' data formats and transport protocols. Mainframe systems are simply at the top of the food-chain.

Most other enterprise systems are being developed based on one of three fiercely competing technology sets (which are the other camps): the Common Object Request Broker Architecture (CORBA) from the Object Management Group (OMG), Java from Sun Microsystems, and the Component Object Model (COM) from Microsoft.

All three technologies are used to *build* enterprise applications, and they or their successors will continue to be predominant choices for that task in the future. However, none of them will ever become the predominant *integration* solution that they were envisioned to be. The single, dominant integration technology will likely be based on XML and simple, open protocols like SOAP that any software vendor can easily implement—and which leverage the equally open Internet infrastructure and protocol suites.

To underline this point, let's take a quick look at CORBA, Java, and COM.

The Common Object Request Broker Architecture

The Common Object Request Broker Architecture (CORBA) was crafted in the early 1990s and has more recently been amended with the Internet Inter-Orb Protocol (IIOP). The CORBA specification is a joint effort by a consortium of well over 300 companies (the Object Management Group) to establish a standard for remotable object implementations.

The CORBA architecture has successfully provided interoperability and infrastructure for object-based solutions in many industries. CORBA object request brokers (ORBs) are popular on UNIX systems because this operating environment does not have a native object model like the Windows operating system family. Hence, ORBs fulfill much the same role in UNIX or in heterogeneous environments as COM does in the Windows

world. CORBA-based systems have been successfully implemented in many industries—from trade to aerospace—and do especially well wherever object-oriented systems with many "live" objects need to be tightly integrated with each other.

However, CORBA suffers the same fate as many architecture-focused approaches defined by industry consortia: it has long lacked full standardization in many of its core elements, leaving fundamental aspects to the software vendors implementing the specification. This problem creates many incompatibilities, especially on the server-implementation side. Any CORBA-based application is locked to the object request broker's vendor and that product's specific enhanced features that fill the void left undefined in the core specification.

In addition, CORBA is extremely complex. The specification effort is driven by many cooperating organizations; however, only a small number of companies are actually capable of implementing a reasonably complete CORBA infrastructure form. Because the investments to create reliable CORBA implementations are substantial and most vendors' target markets are rather small, CORBA products are typically quite expensive.

On the other hand, affordable and even free CORBA implementations do exist. However, they also come with support options that are equivalent to their pricing. And that is usually not acceptable for systems on which you want to run the lifeline of your business.

CORBA's standardized wire protocol is IIOP. While you can implement IIOP independently of a compliant ORB product, it still exhibits most of the complexity issues of the core specification, making it similarly difficult to implement and understand.

The large number of companies supporting CORBA and the number of products implementing the specification appear to confirm the success of that initiative. In reality, the implementations are not necessarily compatible where it matters: on the server. Consequently, your code is locked into a single vendor's product and the platform support is limited to the platform support of that vendor.

Speaking of platforms and CORBA, it's time to move on to the third camp: Java.

Java

For many people, Sun Microsystem's Java is the poster child of the "open systems" idea. While the Java programming language is its most visible part, the Java Virtual Machine (JVM) is the heart of this technology. JVM is a software-based virtual CPU that executes a special command-set, called *bytecode.* JVM is designed to be implemented on top of existing platforms; therefore, Java programs can theoretically be moved between those platforms without changes or, as Sun puts it, "write once, run everywhere."

In reality, the quality of Java applications heavily depends on that of the underlying virtual machine and its capabilities. In addition, the Java model is built on the vision that all services available to a Java application are also written in Java and are therefore agile between platforms. Herein lies the problem. To leverage the power and features of the underlying operating environment, like the high volume transaction capabilities of IBM's OS/390 or the scalability of Sun's own Solaris platform, Java applications need to make direct calls into the underlying platform using Java's raw native interface (RNI) bridging technology. Once you go down this path, the platform agility is lost and the platform abstraction layer with its processing overhead becomes more of a hindrance than an advantage.

For EAI tasks and remote access of objects, Java has its own wire protocol, called Remote Method Invocation (RMI). It's not much of a surprise that RMI is just as complex as IIOP or Microsoft's COM wire protocol (discussed in the following section). RMI is also bound to the JVM standards, such as the Java type system; therefore, it is a Java-centric technology.

In addition, RMI is a core part of the Java standard—as such, it is subject to the standards exclusively defined by Sun and their restrictive testing rules. Vendors cannot compatibly enhance or improve RMI without risking their license being revoked by Sun. In the Java world, Sun decides what is Java, they decide who implements Java, and they ultimately decide who they drag to court if a company decides to improve Java beyond Sun's vision.

The Component Object Model

The Microsoft Component Object Model (COM) was created in 1992 as the fundamental technology for Microsoft's document integration technology, Object Linking and Embedding (OLE). Since it hit the market in its finalized version in 1993, COM has undergone a lot of changes—many of them changes of name. You may know COM under its cover names OLE, OCX, ActiveX, DCOM, or COM+. Microsoft marketing is really good at obfuscating the roots of a technology by giving it a new name on every day it's not raining in Redmond—which, as people say, is a major event in the Seattle area.

From a developer perspective, the fundamental principles of COM haven't really changed since then. But under the hood, the COM functionality and the services available to COM components have grown and improved so much that it became the most commercially successful component and object technology on the market today. However, COM suffers from many of the same limitations as CORBA.

Like CORBA, COM is too complex to implement independently. In fact, while COM's wire protocol has been openly documented as an Internet draft, only a single implementation has ever gotten anywhere near commercial quality, and it even used the Microsoft source code as the starting point: Software AG's DCOM for UNIX, called EntireX. The little-known Microsoft product COM for Solaris is essentially a Microsoft-branded version of the Software AG porting effort. However, none of these ports really did have any substantial commercial success on non-Microsoft platforms.

While the vendor lock-in issue is not obvious in the CORBA space, it is clearly visible with COM. If you commit to COM, you commit to one of the Microsoft platforms. This also means that if you commit to COM, you can talk only to Microsoft platforms.

EDI MEETS XML MEETS EAI

Now that you know the history of EDI and XML and understand the problems with the most widely used enterprise integration technologies, you can clearly see that XML has striking advantages over EDI and the most common EAI technologies. XML-based data exchange protocols are obviously better suited to integrate solutions across organization and platform boundaries than any of the 1990s favorites.

The simplicity and openness of XML is the key factor.

Application integration either of the CORBA, Java, or COM integration models works only if all potential peers can be expected to support that technology in both implementation model and wire protocol. In reality, this was always a very optimistic, marketing-driven wish at best.

The adoption of EDI found its natural barriers in IT budgets. The most widely used EDI standards are too complicated and costly to implement and maintain on a large scale and, at the same time, too static because they do not allow businesses to adjust to rapidly changing needs without violating the narrowly defined standards.

Through simplicity and openness, XML enables broader adoption of electronic data exchange by making it less expensive. XML also allows tighter integration of systems across all platforms. It is simple enough to be directly implemented on any platform and any device, and it is agnostic to all the established rivaling camps.

If you are a developer who has grown up on Microsoft technologies and you haven't been involved in EDI projects (which is the case even for a lot of seasoned software architects), you are unlikely to be familiar with the terminology and complexity that BizTalk brings to the Windows platform from the EDI space. If you have developed EDI applications and are now eyeing BizTalk as an alternative to your current infrastructure, you will be confronted with a lot of Windows-specific terminology and acronyms, like COM, ASP, IIS, or WSH.

Maybe you are confused about BizTalk after browsing the available marketing material. If so, join the club of developers who come from a different background, speaking a different language, and who now meet to build solutions by using a product that bridges the gap between both worlds. Now that you understand the roots of both EDI and XML, you are ready to start your journey of Microsoft BizTalk Server.