Best of Both Worlds: Java & .NET for Fun & Profit

Whitepaper by Ted Neward, Spring 2007, InfoQ.com/j+n

For almost a half-decade now, since the release of Microsoft's .NET Framework, as one of those few "experts" fluent in both the Java/J2EE and .NET platforms, I've been speaking on the topic of Java/.NET interoperability. And regardless of the venue or the audience, one question (from friends, attendees and consulting clients alike) continues to appear at the top of the "Frequently Asked Questions" list for this topic:

> *Q: So… be honest, I promise I won't tell anybody: Which one do you like best? Java or .NET?*

I don't have a favorite; I love them both the same. To be honest, it's not an entirely truthful answer, so it's time to come clean, and go on the record as to which one I prefer.

> *A: It depends.*

The changing nature of the question

A deep divide has fallen across our industry around the basic question of "Which platform do you use? Are you a Java developer, or a .NET developer?" By the tone of some of the discussions held on this topic, one might think this is the major discussion topic of the day, complete with flaring tempers and heated discourse to match. Forget the classic debates of "eminent domain" vs "imperial aggression", or those issues the mainstream media thinks important, like the growing instability in Iraq or the Horn of Africa—if we measure the emotional energy involved, clearly the world's first and most important issue is that of whether you spend the majority of your programming time in Eclipse or Visual Studio.

The truly ironic thing about these debates, interestingly enough, is that they're entirely pointless—Java and .NET, while strikingly similar on several points, are in fact two entirely distinct and different platforms, each with their corresponding strengths and weaknesses. Each platform developed (or evolved) in accordance with the community and culture around it, and as such, each platform looks to solve different problems, in different ways, using different approaches.

What's more, the platforms themselves have begun to diverge in recent years. At conferences, I used to be able to say that the choice between Java and .NET was largely a cultural one, that "anything you could do with one can be done by the other in about the same amount of work". Not so, anymore. While it was fair to characterize .NET 1.0/1.1 as a fairly straightforward across-the-board equivalent to Java, the two have each started to chart differing paths forward, based both on their own unique innovation as well as the reaction of the users utilizing them. The Java community's recent interest in incorporating more dynamism through the language and platform, for example, measured against Microsoft's recent release of .NET 3.0, is a largely apples-to-oranges comparison.

As a result, the question regarding Java and .NET has begun to change subtly; no longer is it "Which platform do you prefer?", but the more interesting—and powerful—question "How can I use each of these two platforms together?"

While a full listing of all the possible integration scenarios between these two incredibly rich platforms is beyond the scope of this paper, we can examine a few compelling ideas, and explore them both in concept and in code.

Scenario: WPF to WCF to Java Web Service

Probably the most common example offered of Java/.NET interoperability is the ubiquitous Web service, typically with a Windows Presentation Foundation or WinForms front-end, using Windows Communication Foundation to do the actual work of transferring the data to the Java Web Service waiting on the other end, typically hosted in some kind of Java container, be it WebLogic or WebSphere or Spring or Tomcat or something similar. The pains and pleasures of building Web services are well-documented elsewhere, so it serves no purpose to repeat them in detail here; suffice it to say, however, that treating Web services as an extension of CORBA or .NET Remoting (that is to say, as a distributed object technology) will generally lead to greater work and effort than is necessary. Services, when used properly, create looser coupling than the distributed object toolkits they seek to replace, specifically to make it easier to pass across technology boundaries such as the one we're discussing. Both WCF and JAX-WS have been written with the notion of "passing messages, not objects" at their core, despite the surface-level APIs that would make them seem more like RMI or .NET Remoting, making each a good choice for building Web services that will interoperate well.

The obvious advantage of this scenario is that each technology focuses on the parts that it does well: the front-end is delivered via a technology that is particular to the platform and can thus take full advantage of its capabilities, and the back-end is written in a platform that has earned a reputation for performance and scalability.

Scenario: SQL Server Service Broker & JSP

With the release of SQL Server 2005 came a new messaging implementation, SQL Server Service Broker, to use in building message-based communication applications. Implemented on top of SQL Server's database engine (the queues in Service Broker are effectively tables with a thin veneer on top of them) and taking full advantage of that robustness to provide transactional and ordered delivery guarantees, Service Broker offers developers a compelling messaging platform, particularly in those data centers where a database is already present.

Accessing Service Broker from Java, however, is not that much more difficult than any other sort of JDBC-based access against SQL Server. A Java application—be it a client app or another server-based processing engine—can access Service Broker through the Microsoft SQL Server 2005 JDBC driver (available for free download from MSDN) and either send messages to a Service Broker service, or receive messages from a Service Broker service, as necessary.

In this example, a fictitious apartment complex wants to Web-enable the generation of work orders for its maintenance personnel, so that renters needn't call the office to place a ticket (and thereby take up valuable office personnel time filling out paper forms in triplicate; office personnel have enough of a hard time ignoring tenants as it is).

As such, the solution provider has built a very simple and lightweight Web-based system with two JSP pages: one for renters to place tickets into the service, and a second for maintenance personnel to gather the tickets up and view them. The intent of the system is simple: the first JSP form takes the ticket information, such as the description of the problem, the apartment itself, the tenant's name and phone #, and so on, and queues that information into a ServiceBroker queue, where it resides until Maintenance staff access the second JSP form to get a list of pending work to be done.

Speaking to the implementation, in many respects, from the Java perspective, working with ServiceBroker is not much different from working with any other JDBC-fronted database; to put messages into the queue requires only a JDBC call into the SQL Server instance, much as a traditional INSERT or UPDATE would be written:

```
Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
Connection conn =
  DriverManager.getConnection(
    "jdbc:sqlserver://localhost\\SQLEXPRESS;" +
    "databaseName=ServiceBrokerExample",
    "sa", "password");

Statement stmt = conn.createStatement();
String sql =
  "DECLARE @conversationID UNIQUEIDENTIFIER; " +
  "BEGIN DIALOG CONVERSATION @conversationID " +
  "FROM SERVICE InspectionService " +
  "TO SERVICE 'CentralMaintenanceService';" +
  "SEND ON CONVERSATION @conversationID ('" +
    workRequestMessage + "');";

stmt.executeUpdate(sql);
```

Fetching a message from the queue is similarly straightforward, using the SQL Server RECEIVE keyword:

```
Statement stmt = conn.createStatement();
String sql =
  "RECEIVE CAST(message_body AS XML)," +
  " conversation_handle FROM" +
  " CentralMaintenanceQueue";
ResultSet rs = stmt.executeQuery(sql);
while (rs.next())
{
  out.write("Message: " + rs.getString(1) + "<br/>");
}
rs.close();
stmt.close();
```

A reasonable question would center around the use of SQL Server's Service Broker here, instead of a more Java-friendly JMS implementation, such as the open-source ActiveMQ or commercial SonicMQ implementations. While it would be easy to fall back on the usual Java/.NET interop answer, "We do it because we have to", there's a more compelling reason here: conversations.

ServiceBroker provides a new feature as yet unseen in the JMS specification, that of the "conversation": similar in some ways to transacted message delivery, a conversation represents a sequence of messages back and forth, and carries a unique identifier for each conversation. In essence, it's a halfway point between a flurry of RPC calls and independent, individually-tracked messages. It provides for a degree of reliability and robustness not typically found in messaged communication systems. Although in our fictitious example above, the use of conversation is somewhat arbitrary, it can be particularly powerful in longer-running business processes. The conversationId identifier, in the code above, is unique across the ServiceBroker instance, and identifies this collection of messages (just one, in this case) for this particular user interaction.

Another reasonable question would center around the use of JSP as the web front-end in place of ASP.NET; again, while it would be tempting to simply cite a "have to" reason such as using non-Windows platform to host the Web tier, JSP offers a compelling reason in its own right, in that there is a wealth of tools and prebuilt componentry for producing nice-looking Web applications. If we extend the discussion to all of the Java/Web space, tools like Struts, Seam, WebWork, JSF, Google Web Toolkit, and more make the Web development experience distinctive from the traditional drag-and-drop approach offered up by ASP.NET. (While drag-and-drop may work for inexperienced Web developers, practiced Web designers have usually found their own approaches they prefer, and find that ASP.NET's design practices clash with their own.)

For a more detailed discussion of SQL Server Service Broker, please see "A Developer's Guide to SQL Server 2005" by Beauchemin and Sullivan. For a more detailed discussion of Servlets and JSP, see "Java Servlets and Java Server Pages" by Jayson Faulkner and Kevin Jones. For a more detailed discussion of JDBC, see "The JDBC Tutorial and API Reference, Third Edition", by Fisher, Ellis and Bruce.

Scenario: Office & Spring

Though it may pain some of the more zealous open-source advocates to hear this, Microsoft Office represents, without a doubt, the world's most popular office productivity suite over the last decade. In many respects, it is the most-installed piece of software in the world, second perhaps only to Windows itself.

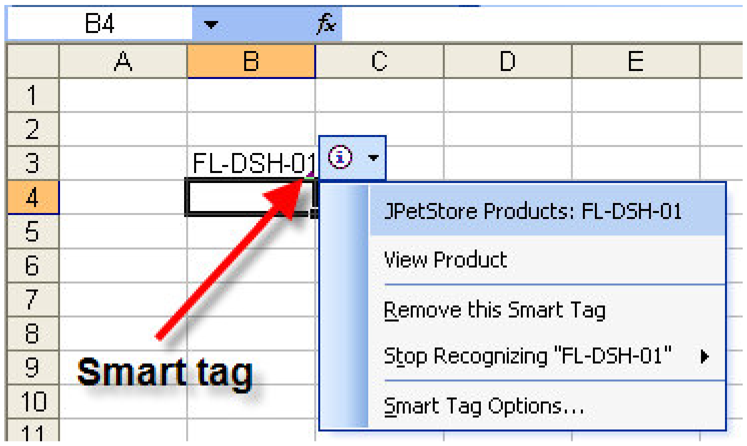
For a few years now, the Java community has discussed "richer" client applications, moving away from the click-wait-read cycle of systems built around the Web and towards a more interactive style of user interface. AJAX certainly enables some of this, at the (sometimes prohibitive) cost of having to write potentially complex scripting code to deal with different browsers and browser versions. Some in the Java community have posited the Eclipse Rich Client Platform as a solution, others push JavaWebStart, or Adobe Flex, and so on.

The best rich client is the one based on the software already pre-installed on the end-user's machine. Given that Office is almost always preinstalled, particularly on machines within a corporate environment, why not use the incredible extensibility interfaces in Office, and use Office as the rich client, with Java as the back end?

Whole forests have been clear-cut in order to produce the myriad books, papers, tutorials and reference documentation on the Office object model and how to use it, both from .NET and unmanaged COM, so duplicating that information here would be counterproductive. Instead, this paper will focus on a single part of Office's extensibility model, that of the Smart Tag, and in particular, the Smart Tag List, a predefined Smart Tag that uses an XML definition file to recognize text in an Office document (typically Excel or Word, though PowerPoint and Access are also able to use Smart Tag Lists) and offer a small drop-down menu that will lead users off to a Web site.

In this case, the fictitious scenario is simple: an online e-tailer has found their online pet shop to be wildly successful (having finally solved the problem of shipping pets through surface mail by negotiating deals with local pet shops around the world), and their portal, based on the Spring JPetStore example, now needs to handle all sorts of complex calculations and business rules as defined by the accountants and marketers within the company. The simple orders are easily left to the portal, but more complex orders will be handled by salespeople, either in person or over the phone.

Complex calculation rules demand a complex processing language to handle them, and this is exactly the kind of scenario that Excel was created for—in fact, both the accountants and marketers can write the rules in Excel's formula language themselves—and so we want to take the next step of enabling the Excel spreadsheet to act as the front-end to the Spring portal. In this case, the first step is simply to recognize the order and product numbers in the Excel document, and display a Smart Tag that takes the salesperson over to the appropriate page on the Spring-powered Website. (Future enhancements could automatically place the order when the spreadsheet is saved, or pop warning messages when trying to sell pets that the store is currently out of, and so on.)



Doing this is actually more an exercise in writing a simple XML file than it is in writing Java or .NET code; thanks to the flexible nature of URLs, the smart tag list can remain blissfully unaware of the fact that the website behind the URL is implemented in Spring. The Smart Tag List document, shown below, even refreshes itself every day, on the grounds that new product IDs may come available ("Look, kids, we now stock ferrets!").

```
<FL:smarttaglist
xmlns:FL="http://schemas.microsoft.com/office/smarttags/2003/mostl">
<FL:name>JPetStore</FL:name>
<FL:lcid>1033,0</FL:lcid>
<FL:description>A list of JPetstore symbols for recognition, as well as a
set of actions that work with them.</FL:description>
<FL:moreinfourl>http://localhost:8080/jpetstore/shop/index.do</FL:moreinf
ourl>
<FL:updateable>false</FL:updateable>
<FL:autoUpdate>false</FL:autoUpdate>
<FL:lastCheckpoint>400</FL:lastCheckpoint>
<FL:updateURL>http://localhost:8080/jpetstore/jpetstore-smarttag-
update.jsp</FL:updateURL>
<FL:updateFrequency>5</FL:updateFrequency>
<FL:smarttag type="urn:schemas-tedneward-com:office:smarttags#jpetstore-
products">
<FL:caption>JPetStore Products</FL:caption>
<FL:terms>
<FL:termlist>
FL-DSH-01, FL-DLH-02, FI-FW-01, FI-FW-02, RP-LI-02, RP-SN-01
</FL:termlist>
</FL:terms>
<FL:actions>
<FL:action id="urn:schemas-tedneward-com:office:smarttags#jpetstore-
products:View">
<FL:caption>View Product</FL:caption>
<FL:url>http://localhost:8080/jpetstore/shop/viewProduct.do?productId={TE
XT}</FL:url>
</FL:action>
</FL:actions>
</FL:smarttag>
<FL:smarttag  type="urn:schemas-tedneward-com:office:smarttags#jpetstore-
items">
<FL:caption>JPetStore Items</FL:caption>
<FL:terms>
<FL:termlist>
EST-16, EST-17, EST-6, EST-7
</FL:termlist>
</FL:terms>
<FL:actions>
<FL:action id="urn:schemas-tedneward-com:office:smarttags#jpetstore-
items:View">
<FL:caption>View Item</FL:caption>
<FL:url>http://localhost:8080/jpetstore/shop/viewItem.do?itemId={TEXT}</F
L:url>
</FL:action>
</FL:actions>
</FL:smarttag>
</FL:smarttaglist>
```

Picking this apart briefly, we're essentially setting up two smart tags, one to recognize the Product IDs (FL-DSH-01, and so forth), and a second to recognize Item IDs (EST-16, EST-17, etc). In each case, we simply surf over to the website, passing the ID in place of the {TEXT} placeholder in the URL. Here, the IDs are hardcoded, but notice how the <updateURL> tag lists a .jsp page—the JSP code there queries the underlying database for all Product and Item IDs and lists them out when it sends back a new copy of this Smart Tag List document (which Office will silently copy over the original, located in the C:\Program Files\Common Files\Microsoft Shared\Smart Tag\Lists directory). Office knows to update this Smart Tag List every 5 minutes, because the Smart Tag List defines itself to be updateable (as given by both the <updateable> and <autoupdate> tags above), and that it should query for an update every 5 minutes (as given by the <updateFrequency> tag). This means that, silently, the smart tag will update itself as new products and items are introduced into the database, without any manual user intervention required.

Smart tags are far more powerful than this simple example leads us to believe; the Visual Studio Tools for Office API allows the .NET developer to write any sort of code behind a smart tag desired, so it's not infeasible to imagine issuing a remote call (whether a Web Service call or through a commercial toolkit, such as JNBridge or ZeroC's ICE) to the JPetStore engine to obtain current inventory counts at the time of the smart tag's activation, and so on.

Additionally, smart tags are hardly the end of Office's integration capabilities; the document pane can be customized to provide another user interface into any Java system, Excel's formula language can be extended by custom formulae (which, of course, could either host the JVM locally to make use of Java APIs or else call out to Java systems to do the same), and so on. And this need not all go one way—if desired, Word or Excel itself can be hosted inside of the Eclipse RCP, as can any COM Automation object, where all of the features of Word and Excel will still remain available.

Other Scenarios

Certainly, these aren't the only scenarios possible, just the few that came to mind during recent discussions and client meetings. Other scenarios include:

- *PowerShell using Cmdlets that speak to Java.* PowerShell is poised to become the most important administration tool for Windows for the near future, and it would be a relatively trivial exercise to build a set of Cmdlets that interrogated Java servers using JMX. This would make it possible—simple, even—to build scripts that checked both the status and performance of IIS- and Java-based servers with a single script, commingling the results into a nice graph (such as those produced by the cmdlets from PowerGadgets), or to be able to turn on and off various parts of the system via method calls.
- *Java using Speech Server.* Vista has some new speech synthesis capabilities, and Microsoft's Speech Server offers some powerful speech-analysis capabilities that currently don't exist in the Java platform. As we become more aware of physical disabilities in our users, speech and interacting with users through voice (whether over a phone or through a microphone in front of the computer) becomes more and more attractive.
- *Workflow activities calling Java.* Windows Workflow has a prebuilt activity that calls out to a Web service already, but, as mentioned earlier, Web services are useful under certain circumstances, but are hardly a panacea for all interoperability tasks. Custom activities could make use of other Java/.NET interoperability approaches to talk to Java components.
- *Java hosting Workflow.* One of the most powerful facets of the Workflow engine is its ability to be hosted in a variety of environments, such as ASP.NET. Certainly, there's no reason why the Workflow engine couldn't be hosted inside of a Java process, such as Tomcat or Jetty, thus enabling Workflow's "information worker" accessibility to reach out to both Java and .NET-based web applications.
- *Windows Mobile devices interoperating with Java.* As the mobile device world heats up, Microsoft's Windows Mobile platform stands as a viable platform for writing software to run on mobile devices, such as the Smartphone. As these devices become more ubiquitous, it's natural that IT departments will want to integrate them into their already-heterogeneous environments, which means Java will likely be involved. Sometimes this communication will be over Web services, but in some situations a more focused communication method will be necessary, such as using a proprietary toolkit like JNBridge Pro or ZeroC's ICE.

As more and more developers come to realize the power of using both .NET and Java together, more scenarios will likely come to light. And as both the Java and .NET communities come out with more innovative ideas, these will create even more reasons for each side to openly and honestly consider how to use the other to best solve our clients' problems. Because, after all, in the end, regardless of which technology you love more, that's what we're about: providing solutions to our clients.

About the Author

Ted Neward is the principal of Neward & Associates, a consulting group that focuses on enterprise systems using Java, .NET, XML, and other tools as necessary. He has been using C++ since 1991, Java since 1997, and .NET since 2000. He is a .NET instructor with PluralSight, teaches Java independently, speaks at conferences worldwide in both the Java and .NET communities, writes for MSDN, InfoQ and TheServerSide, authored the books *C# In a Nutshell*, *SSCLI Essentials* and *Effective Enterprise Java,* among others, and can be found on the Web at http://www.tedneward.com.

About InfoQ.com's Java + .NET Integration Portal

Java and .NET represent the extensive share of enterprise development. At http://infoq.com/j+n we are hosting and continually posting that will help you learn how you can leverage the strengths of each together, such as using Microsoft Office to act as a "rich client" to a Java middle-tier service, or building a Windows Presentation Foundation GUI on top of Java POJOs, or even how to execute Java Enterprise/J2EE functionality from within a Windows Workflow host. This whitepaper was produced for the InfoQ Java + .NET portal.

Appendix: Dramatis Personae

It's important to acknowledge that readers of this paper will generally have experience and knowledge of one of the two sides, not both. For that reason, a laundry list of the major components of both platforms appears below. This isn't intended as any sort of overview of explanation of those components, nor is it an exhaustive list; readers are encouraged to consult the Bibliography at the end of this paper for more on each topic.

Java:
- *Java 5 Enterprise Edition.* Recently renamed from its former moniker "Java2 Enterprise Edition" and still commonly referred to as "J2EE", this specification is an umbrella specification, bringing together dozens of other "enterprise-scope" specifications. Although incorrect, many use the term "J2EE" as synonymous to "EJB".
- *Enterprise JavaBeans (3.0).* EJB is a specification describing a container into which software components seeking lifecycle, connection and distributed transaction management are deployed. It is fair to characterize EJB as the logical Java successor to transaction-processing mainframe systems.
- *JDBC (4.0).* The Java standard call-level interface API to relational database implementations. Different vendors provide different "providers" (drivers) which implement the JDBC API, thus allowing the programmer to remain ignorant (and, theoretically, loosely-coupled) to the actual database implementation.
- *Servlets (2.5).* The Servlet specification describes a container into which software components designed to build dynamic HTTP/web pages are deployed. A Servlet is essentially a Java class extending a particular interface.
- *Java Server Pages (2.2).* JSP pages are an output-oriented way to create servlets, similar in the way that ASP or ColdFusion pages look. JSP files are then translated into Java source (servlets) and compiled.
- *Remote Method Invocation.* RMI is Java's object remote-procedure-call (ORPC) stack. RMI has two flavors, one using a native Java wire format, called "RMI/JRMP", and the other using

OMG's CORBA wire format, called "RMI/IIOP". Officially, J2EE systems are encouraged to use RMI/IIOP, but in practice RMI/JRMP use is more widespread.

- *Java Message Service (1.1).* JMS is an API for standard access to any messaging service (not to be confused with email service) for the Java platform.
- *JavaMail.* JavaMail is an API for standard access to any email (SMTP, POP3, IMAP, and so on) service.
- *Java Naming and Directory Interface.* The standard Java API for any service that provides naming and/or directory services, such as LDAP.
- *Java WebStart.* A deployment technology where applications can be launched locally from an HTTP URL, and stored on the client machine for future (offline if desired) execution.
- *Java API for XML Binding (2.0).* JAXB is the standard API for automated Java-to-XML/XML-to-Java transformation.
- *Java API for Web Services (2.0).* JAXWS is the standard API for Java XML-based services. Originally, JAXWS was called the "Java API for XML RPC (JAX-RPC)", but that name was deprecated in the 2.0 release as JAXWS incorporates a more message-oriented approach.
- *Spring (2.0).* A *de facto* standard open-source container providing lighter-weight services to Java components (also known as "POJOs", short for "Plain Old Java Objects"). Widely considered the replacement for J2EE.
- *Swing.* Officially known as "Java Foundation Classes", Swing is a cross-platform user interface toolkit for building rich-client UIs. Because it seeks to create visual consistency across platforms, Swing implements most of its own painting and display logic.
- *Standard Widget Toolkit.* The UI technology at the heart of the open-source Eclipse IDE, SWT is another UI toolkit, different from Swing in that it relies on native OS-level UI facilities to do its painting and display logic.

.NET:

- *Windows Communication Foundation.* Once code-named "Indigo", WCF represents Microsoft's next-generation API for doing any sort of program-to-program communication, from message queuing to secure/reliable/transacted services to WS-* web services.
- *Windows Presentation Foundation.* Once code-named "Avalon", WPF represents Microsoft's next-generation presentation layer, looking to take advantage of the huge hardware investments the industry has made in graphics cards over the years. WPF code can be used in two forms, either called-and-compiled as per normal .NET development, or written declaratively using an XML dialect called "XML Application Markup Language" (XAML) that can either be compiled into the application or sent over HTTP requests to IE 7 browsers for dynamic display. A subset of WPF, called WPF/E, has been released for use by non-IE browsers.
- *Windows Workflow Foundation.* Workflow, as it's commonly called, provides
- *Windows Forms.* The .NET wrapper around the traditional Windows UI facilities (User32.dll and GDI32.dll).
- *Active Directory.* AD is a directory service intended for enterprise-wide deployment of "named resources", such as users and servers. AD also comes in a lighter-weight version for per-application use called "ADAM"
- *ASP.NET.* The .NET implementation for creating dynamic Web/HTTP facilities. The ASP.NET pipeline provides both programmatic (ASHX) and output-oriented (ASPX) forms for producing end-user visual content, as well as programmatic Web services (ASMX).
- *ADO.NET.* The call-level interface API to relational database implementations. Different vendors provide different "providers" (drivers) which implement the ADO.NET API, thus allowing the programmer to remain ignorant (and, theoretically, loosely-coupled) to the actual database implementation.
- *.NET Remoting.* .NET's object remote-procedure call (ORPC) technology.

- *Microsoft Message Queue (4.0).* MSMQ is Microsoft's messaging service, available for all recent versions of Windows (4.0 ships with Vista).
- *COM+/Enterprise Services.* COM+ is the container providing transaction and lifecycle services into which "managed applications" (as they were originally known) are deployed. .NET components use COM+ through the System.EnterpriseServices namespace.
- *Microsoft Office.* The world's most widely-installed office-productivity suite, its principal parts consist primarily of Microsoft Word, Microsoft Excel, Microsoft PowerPoint and Microsoft Outlook.