Eden
Powering IT & Engineering

The Impact of Microsoft.NET on the ERP Market

Compiled By: Team dot net [Eden IT Services Division]

Introduction

A low rumble is emanating from Microsoft about its new .NET initiative. The volume is rising. Have you heard it? You will. .NET (pronounced "Dot Net") will be ringing in our ears by fall 2001 when the first .NET software is officially released and **Microsoft®** brings its marketing machine up to full power.

.NET is a mammoth undertaking by the world's largest software company. .NET will impact all users of Microsoft products over the next few years as it ushers in a new wave of software innovation. Unfortunately, like many of the Microsoft technical initiatives before it, the shift in core thinking that is central to .NET may become lost in the marketing hype. As the various Microsoft divisions jump on the bandwagon, everything at Microsoft will quickly become .NET-this and .NET-that. This dilution of the message has already begun with Microsoft's release of new .NET versions of its family of enterprise server products. As soon as a Microsoft product becomes friendly with Extensible Markup Language (XML), it seems to get the .NET label.

.NET is much more than adding XML support. Before the importance of .NET is clouded by all the marketing hoopla, let's examine the core concepts of .NET and what .NET means for manufacturing companies. This article will explore the roots of .NET, give a quick overview of .NET technology and conclude with a discussion of how the changes .NET portends will be particularly interesting for manufacturing companies.

The purpose of .NET

In the mid-1990's, it became clear to Microsoft that the Internet would change the computing landscape in a big way. While the public was being dazzled by new websites and the promise of dot-com business plans, the architects at Microsoft were envisioning the next generation of Internet usage. As they saw it, software that operates independently on individual computers would give way to a future filled with cooperating applications running in a rich soup of web-based services. Microsoft formed a grand plan to rework the core languages and tools used to develop Windows software. Microsoft aimed to leverage Windows to stake the high ground in the new computing landscape.

.NET is based on recognition that **the Internet is the application platform of the future**. This statement can be viewed as the underlying goal of .NET, the root cause of all the commotion. On the surface, this may not seem like such a big deal, but for Microsoft, it changes everything. For years, Microsoft has been adding Internet-related features to its products. But these changes have really only been skin-deep. To attain the new .NET goal, Microsoft products must evolve to actually **run on the Internet**. Microsoft toolsets and languages must make it easy to write new software that leverages the new 'connected' world.

These same Microsoft architects also recognized that there were a lot of things they really didn't like about Windows and Windows software development. So, they set out to fix it. Fix it all. As we will see, .NET is a grand undertaking.

NET addresses many problems

To understand what .NET is, let's look at the problems that Microsoft set out to solve.

Continue the course

First let's get the usual boilerplate goals out of the way. A lot of computing experts will preach that in the future, applications will be structured to work more as services. For instance, users may soon be expected to rent the use of a spreadsheet or word processing service over the Internet rather than to rent the actual software and load it on our local PC as we do today under the current licensing scheme. This new 'software as services' model is coming.

Experts will also tell you that our computing experience will continue to evolve to become more 'document-centric' rather than 'application-centric'. This follows the idea that you shouldn't have to care so much about what software you are running. You should only need to focus on the information, the document you are working on, the essence of your job/task. Software should be ever-present, delivering service for those who pay the fee, but reside in the background – like your electric company, or your cable company.

But we've heard about these goals before. The computing experience is evolving and these are two of the commonly accepted directions. .NET is certainly an enabler of these changes. But the real exciting aspects of .NET are the other new goals Microsoft set for itself.

Fix what's wrong with Windows

Where .NET starts to get interesting is that it is aimed, in part, at fixing all those things we've come to hate about Windows.

Let's start by talking about the process of installing software. It's a given that the installation of new Windows software has the potential to corrupt previously installed software. The root cause of this problem is that sharing software components is the rule - a rule laid down in Building 2 at Microsoft long, long ago. This rule was born when computing resources were scarce. What do we do when something is scarce? We economize, we conserve. To early Windows designers, this meant that applications should share software code whenever possible. There would be less code in total, which would take up less space – both memory and disk. This must have seemed like a nice plan at the time, but we've all been paying the price ever since. When the installation of a new application 'upgrades' a shared piece of software to a version that no longer works with the other software packages on our computer we get to experience what has come to be known as "DLL Hell". To make matters worse, all of this sharing is orchestrated by a deviously complex and delicate mechanism – the Windows Registry. When was the last time you installed new software without worrying that you were about to break something near and dear to you? And to venture into the Registry to attempt to correct a problem is to take your computer's life in your (unqualified) hands. Let's face it – Windows is fragile.

While installing new software is scary, removing software is also a challenge. Each new software package smears itself all over your hard drive. When is the last time you uninstalled some Windows software and truly felt like you removed it all? It's just plain difficult keep a clean house. Eventually our disk drives fill up with trash left behind by old applications, old versions, and old demos. Windows is basically messy.

Another complaint about Windows applications is that we can't quickly move an application from one computer to another. You can't just relocate the darn things; you've got to completely re-install each software package on your new system. And it's likely you'll have to re-make any personal configuration changes or fine-tuning you may have done to get things to work the way you want. Do you know the secret to moving your e-mail address book over to your new laptop? It shouldn't have to be this hard.

Eliminate the dilemma

One of the biggest attractions of the Internet has been the ease of using web browser-based applications since there is really no software to install. You just launch your browser and click from web page to web page and do your thing. In certain cases, some local software is needed, but it is usually downloaded and installed for you automatically. As a result of this experience, "web deployment" and "light client" have become the mantras of software design teams and IT departments the world over. But there is a catch. Web applications often give up a lot of functionality. Technical folks debate the merits of 'rich applications' versus 'reach applications'.

These terms describe the two ends of the user interface spectrum. A truly rich feature and user experience requires software to be installed and run locally. Conversely, web applications reach a long distance, meaning require no local installation, but provide more primitive features. Lately the 'reach' advocates have been winning most of the debates because ease of deployment has come to be more important to us than a rich feature set. It's a sad situation, when you think about it. Clearly we all want to install and uninstall software easily and we're willing to trade away functionality to get a more carefree experience.

However, even when we accept the watered-down user interfaces, web applications have their own set of shortcomings. When was the last time you tried to clean all that "Internet sludge" off your system? All those cookies, HTML files, and other secret stuff fills up the dark corners of your hard drive. Our systems seem to run slower and slower every month as the crud accumulates.

More and better software

Finally, we users want more features and fewer bugs. Indirectly, we all want software developers to be able to do their jobs quicker and better. Solutions to this problem have been discussed for decades under the headings of "object oriented programming" and "component-based software". Unfortunately, to date these technologies have not really delivered on their promise. Many aspects of the software industry still mirror the pre-industrial age, producing one-off, hand-crafted parts.

Is having software that is simple to install, easily moved, easily removed, runs without bugs, runs without corrupting something else, runs fast, and provides the richest user interface experience too much to ask? No, it's not. Microsoft was listening. In fact, Microsoft was studying these problems long before many of us could even begin to verbalize them.

.NET is a new model for a new world

.NET is clearly Microsoft's solution to fixing the problems of Windows. But before we dive into what .NET is, we need to look at the other big forces shaping our computing world to learn the rest of the purpose behind .NET.

Hardware companies are shaping our world

An irony of the web revolution is that the Internet browser, despite the high technology therein, is still very much like a 'dumb terminal' of the 1960's and 70's. Web applications, for the most part, execute their logic on the web server. Server farms have replaced mainframes. Usually, just the image we see, in the form of an HTML page, is passed across the web. Very little processing power is really needed on our PC to run the browser and display the web page.

As web browsers have become a part of our lives, the chip makers have continued to give us a steady stream of faster and faster processors. Inexpensive memory, the relentless pace of Moore's Law, and continued miniaturization enable ever-growing computing power to arrive each year in ever-smaller packages. It used to be said that the modern automobile has more computing power than the Apollo moon-lender. Nowadays, that can probably also be said about some cell phones. We are racing into a future of ubiquitous computing devices. And we are not talking about wimpy, dumb-terminal, web browsing devices. In coming years we'll have super-powerful hand-held devices or "modular computers". Computers will become almost invisible as their components disappear into our shirt pockets, our eyeglasses, or hang on our belts. Web browsers do not leverage the new high-powered hardware. The advancements in hardware are inviting a new computing model. We will accept the invitation.

Internet companies are shaping our world

Each week we find a richer selection of services available to us on the web. It is now commonplace to check the traffic cameras before you drive downtown, or to peek at the weather through the mountain-top camera before you leave the house to go skiing. Catalogs, theater schedules, maps, stock quotes – we now look to the web for information every day, and often every hour. The Internet provides us with instant access to information that is dynamically changing, as well as the contents of the world's libraries, and much in between. And this is just the tip of the iceberg. It is amazing that it has come so far so fast since most of us can still only receive these services when we sit, tethered to the Internet by copper phone lines. Over the next few years new dot-com economics, made possible by ubiquitous micro-billing systems that can charge a few pennies for a service, will give birth to a flood of new specialized web services for-hire. Services will provide the information we crave. Web-based services will act as agents for each of us, doing our bidding, doing our chores. Services will leverage other services in unimaginable ways. It is all about information and leveraging that information. It's all about more.

Communications companies are shaping our world

It is clear; the future will be filled with extremely smart devices capable of significant computing chores. Our cool new computing and communicating devices will draw information that interests us from a virtual banquet of web-based services. Our personal and business systems will detect, integrate, and aggregate the payload information from these services. We can be confident that, eventually, everything will be connected together by high-speed, broad-band communications, operating over optical fiber, wireless, and laser beams. With the improvements in communications, our devices will communicate constantly, cooperate, and process information for our convenience and benefit, anywhere, anytime. We will be perpetually connected.

Welcome to the wired world. One day, the web browser computing model we use today will simply be a side note, a legacy information publishing mechanism. We will live in a wild new world of fantastically powerful, infinitely connected, collaborative processing.

Introduction to .NET

.NET is Microsoft's comprehensive solution to all of the requirements and opportunities discussed above. .NET redefines the structure of Windows software from the ground up to make Microsoft products a foundation for the wired world. .NET is a direct response to the forces shaping the computer world. .NET is a technology designed to facilitate these changes.

To understand the technology shift that is .NET, you must remember that most modern software systems are comprised of components that cooperate to accomplish the desired result. We are talking about object-oriented (OO) programming here and Microsoft has been using OO techniques for many years to improve the features, robustness, and maintainability of its operating systems and applications.

Take Microsoft Word for example. Word was first written as a monolithic program – a long list of procedures and instructions to accomplish word processing. To combat the bloat as more and more features were added to the program, Microsoft re-created Word using the C++ language and object oriented design principles. This was a painful task, but it was eventually accomplished and Word came to own the word processing market as part of the Microsoft Office suite. Today, Word is actually an aggregate of many, many cooperating objects. Each object can be enhanced more or less independently of the others and new features can be added to individual objects without the whole application becoming too complex for Microsoft to maintain.

The objects that comprise Microsoft applications like Word communicate with one another using a technology called Component Object Model (COM), a standard that Microsoft developed. To date, COM has been a great success story despite the criticism that it is the source of DLL Hell. COM has gotten Microsoft this far, but its days in the limelight are over. You see, COM is not an Internet standard; it is a proprietary protocol. As such, COM-based communications can't pass through

Internet firewalls like HTML pages do. Sure, there is DCOM (Distributed COM), a descendent of COM that allows objects to communicate over networks, but DCOM isn't an Internet standard and can't pass through firewalls either.

In recent years, Microsoft has revised products such as Office, Exchange, and SQL Server to leverage the Internet and Internet technology. Each has been enhanced in various ways: to generate and/or consume HTML pages, interact with web browsers, and lately, to emit and consume XML. But under the covers, these products remain firmly based on COM. Despite the marketing hype to the contrary, Internet support to date is still superficial. To actually run over the Internet and achieve the .NET goals, software objects need to communicate in ways that comply with Internet standards. The complex systems of the future will be comprised of objects distributed around the Internet yet cooperating as the components of Word do today.

Note that in this discussion of software objects, we are not talking about the simple case of a bunch of peer objects. Instead, we are talking about big objects comprised of smaller objects, which in turn are comprised of even smaller objects, etc., etc., down to essentially the software equivalent of atoms. So, when we talk about COM and a replacement for it, we are talking about the very glue that holds the atoms of big applications like Microsoft Word together.

COM is to Microsoft applications and their component objects as air is to humans. We live in air, we Breathe air, and we communicate through air. Likewise, COM objects exist in a COM world – Microsoft Windows operating systems. As we've noted, the Internet is not based on COM. It's based on an entirely different set of standards including HTTP, HTML, and TCP/IP. These Internet standards are as different from COM as air is from water. The analogy I'm trying to make is that the challenge Microsoft has set for it is the software equivalent of converting humans from breathing air to breathing water. This comparison may sound a little goofy, but you should be getting a feeling for the magnitude of the change. .NET is a huge shift. Microsoft has likened the switch to .NET as being bigger than the change from MS-DOS to Windows. I hope you can now begin to see why.

So what will Microsoft use to replace COM? Microsoft will change its objects to communicate using a technology called Simple Object Access Protocol (SOAP) or an evolutionary descendant of SOAP. SOAP is based on XML. XML is essentially just text containing name/value pairs that follow certain structural rules. XML is an Internet standard. Object interactions using SOAP will pass across the Internet and through firewalls - the same firewalls that block COM communications.

Don't get the idea that COM is going away entirely. COM will live on as a legacy standard for objects to use to communicate when they reside on the same Windows computer or can count on

communicating with one another across a Windows network. COM is currently more streamlined and faster than SOAP. But, where the requirement is to communicate across the Internet, COM will be replaced by SOAP. Eventually, increases in hardware speed and network bandwidth will probably eliminate concern for the inefficiencies of SOAP.
To adapt to SOAP and XML, Microsoft must start from the bottom to re-assemble all of its objects using the new glue. To make the transition to .NET, Microsoft had to start with the software equivalent of atoms and molecules and work up from there. The first step of this task is nearing completion. Microsoft has finished re-vamping its languages and compilers and the primitive building blocks that underlie all Windows software. Microsoft will use these new tools and components to rewrite its applications to attain the overall goal for .NET. Microsoft customers can use these same tools to build their own new .NET applications or rebuild their old applications to operate over the Internet or intranets, as necessary. The tools are the part of .NET that is getting the most attention right now and will be the first true .NET software to be released later this year.

What is .NET, really?

.NET is a complete rethink of how software should operate and how distributed applications should be deployed. .NET is a blueprint for building web services and for building systems that leverage web services. .NET is a brand new toolset of languages and compilers. .NET is a bundle of building blocks for the next generation of software applications. In summary, .NET is a kit and instructions for building the Digital Nervous Systems that Bill Gates has been talking about.

The epicenter of the .NET world is a small chunk of exotic software called the Common Language Runtime (CLR) and a bundle of foundation components and basic services called the .NET Framework. Together, the CLR and Framework provide the powerful infrastructure and rich set of built-in features necessary for objects to readily and securely communicate over the Internet. This fall, Microsoft and its partners will release new versions of all the popular computer languages and compilers to support the CLR and Framework. The beauty is that software developers can continue to write software code in their favorite language. The code they write, in whatever language were chosen, and are then compiled by the new .NET compilers into a standard format called Intermediate Language (IL). The IL is then executed at runtime by the CLR. Objects written and compiled in one of these new .NET languages will be able to run on any operating system platform that supports the CLR and Framework.

Microsoft will provide the CLR and Framework for all current Microsoft Windows operating systems. The CLR will be built into future versions of Windows and Microsoft is expected to distribute the CLR as a free CD to retrofit all current Windows systems. Observers expect that the CLR will be adapted for other major operating systems also, including Linux and various flavors of UNIX. Read between the lines. .NET has the capability of setting Microsoft free from the Intel platform. In fact .NET may eventually allow Microsoft applications to transcend operating systems.

.NET and the manufacturing industry

Granted, the task for Microsoft and other software companies will be to recompile their applications using the new .NET tools and components and then enhance their applications to make use of newfound freedom to roam the Internet landscape. Let's consider the more specific benefits that .NET is intended to deliver and how the manufacturing industry in particular will benefit.

Lower cost of system administration

The improvements to software quality and deployment techniques made possible by .NET will mean less IT department expense to install, maintain, and administer Windows software systems. Time spent by IT personnel responding to the current problems of Windows administration will be reduced. IT resources can become less reactive, leaving more time to attend to leveraging technology rather than protecting against it, or recovering from it. Let's look at how these gains will be made.

Install by Copying Files

One of the most exciting aspects of .NET software (software that is written for the CLR) is a feature called 'side by side execution'. With .NET, sharing software components is now the exception rather than the rule. After the developer is able to compile and run the software on his own machine, all that need happen is that the program files be physically copied to the users' machine. If the files arrive there intact, they will run. This is a slight oversimplification because the developer can still share components if they wish, and more complicated installation scenarios can be created. But, the bottom line is that for many .NET applications, the software can be installed simply by copying files onto your disk drive. To delete an application, you need only delete the folder containing the application files. In fact, two versions of the same application can exist, side by side in separate folders. They run side by side – no conflicts, no sharing. You can move an application from one computer to another by just coping it, retaining any configuration settings. Under .NET, applications are now self-contained little bundles. And "little" is another important word. Next, let's talk about that.

Reach and Rich at the Same Time

The CLR and Framework can be thought of as a kit full of all the standard nuts and bolts, valves and pipes, doors and windows that all applications require. By expecting this entire common infrastructure to already be present on the computer, applications no longer have to carry around their own copies of these standard components. Therefore, under .NET applications become much smaller. In fact, in the .NET world, regular Windows applications become so small that they can move across the Internet or an intranet as quickly as web pages. They can be served up like web pages. They land on your system and run locally. And if you don't want to keep one, you can simply delete its folder with confidence that you have deleted the entire application. With .NET, 'rich' user interface applications get legs; 'rich applications' can beat the 'reach applications' at their own game. We have our cake and can eat it too - easy deployment for all! Furthermore, large client-server systems can now employ a 'trickle' install strategy whereby only the components that are needed are actually downloaded to the client – at the moment they are first needed. And .NET provides the means for newer versions to download automatically, too.

These changes will simplify the lives of IT personnel. New .NET applications will be cohesive and lightweight packages that stay put, can be easily installed, clearly seen and managed, and do not interfere with other applications. Time spent troubleshooting Windows software will be minimized. The Windows Registry will become a historical artifact.

Better internal systems through better tools

The powerful new .NET tools will mean easier and cheaper in-house development. Savvy developers can exploit the .NET Framework to tackle bigger challenges and more quickly address custom requirements not satisfied by commercial software packages. New systems built using the .NET framework will be more robust, more powerful, and be delivered with fewer bugs, faster.

If we liken writing a software application to building a house, early programmers were forced to create their own nails and lumber before they could lift a hammer. Later, Microsoft and other companies started to provide standard versions of these basic components. With the .NET Framework Microsoft is now providing the software equivalent of pre-hung doors, trusses, and prefab walls. The .NET Framework is rich with industrial-strength components, born of many years of **Windows software trial and error**.

These prefabricated elements make coding faster and more robust. Rather than write volumes of code, developers can often now just write small snippets of code to connect major components to one another. Fewer lines of specialized code result in fewer places for bugs to hide.

The whole .NET infrastructure was designed to allow applications to make their home on the web and to leverage the web. The .NET tools and Framework will make it much easier for in-house developers to tackle the hard issues of the new connected world. Sophisticated functionality is now built into the CLR and Framework to make creating web services a relative breeze. The .NET Framework provides a rich set of infrastructure elements for security, authentication, identity, notification, and directory services. Developers can concentrate on grand innovation rather than low-level plumbing.

The .NET Framework is compatible with COM, allowing both technologies to coexist and components of each to interact. The result is that IT departments can orchestrate gradual migration of old COM-based software to new .NET technology.

The CLR also has new capabilities for managing code as it runs, providing automatic solutions to old problems like infinite loops and memory leaks – the causes of many a mysterious Windows crash. Code running on the CLR platform promises to be much more robust than the software we are all used to.
.NET features enable new approaches to old information challenges. For instance, the .NET deployment model will make it possible to upgrade running applications, easing the effort to accomplish 24 by 7 operations for many smaller manufacturing organizations.

Better ERP Systems

The advantages discussed above for in-house software development certainly apply to commercial software companies also. Of particular note are the new advances in software speed and the enhanced support for web services offered by .NET. ERP companies who embrace .NET will introduce innovative new features that leverage these advances.

Higher computing speeds will fuel innovation

Over the years, the performance of Windows software has become burdened by many layers of technology translation logic. Old technology has repeatedly been 'wrapped' by new technology code like coats of paint on an old house. In some cases the layers date back to MS-DOS. The .NET project gave Microsoft engineers an opportunity to start over with a clean slate – to scrape away the layers. .NET software is therefore free of the old baggage. It is lean and efficient and promises much higher performance compared to Windows software as we know it.

Also, in building the CLR and Framework, Microsoft brought together its brightest minds to work in concert. Instead of the various language teams at Microsoft each working to solve the same problems in their own way, now all language units put their focus on a single, common infrastructure. The CLR is highly optimized and will continue to get the attention of everyone as it underlies everything. .NET applications will all benefit from the speed improvements made possible by this concentrated focus of talent.

New .NET software will be inherently faster. This speed will be multiplied by new hardware optimized for Internet standards and ever increasing raw processor performance. Dramatic gains in speed will allow developers to completely re-think solutions to business problems.
Improvements in processing speeds will be particularly applicable to manufacturing planning. Old approaches to MRP and CRP concepts will yield to previously unthinkable computing power. In the end, .NET will enable blindingly fast executions of production planning, shop floor tracking, and order handling processes, order promising, dynamic inventory status, and order status analysis. Couple these advances with web services developed using .NET and we will see a new wave of dynamic real-time simultaneous material and capacity planning that may span the boundaries of individual companies, forming integrated planning webs across the supply chain.

Leverage the web services model

The .NET tools and Framework make it much easier to build robust software that emits and consumes web services. Web services can supply instant information in a neutral and self-describing format, listen for information, act as agents, and perform chores. Services can search for, discover, and consume information from other services, aggregate it, digest it, and act on it.

About Eden Information Services

Eden has well established & commendable track record in supporting leading Energy, hydrocarbon, Engineering, Manufacturing industries in their product development support, optimizing their development time & processes. Our unique business model is built around relationships. Our relationship quotient is all about commitment, flexibility and top-to-bottom approach path. Eden has well established & commendable track record in supporting leading Energy, hydrocarbon, Engineering, Manufacturing industries in their product development support, optimizing their development time & processes. Our unique business model is built around relationships. Our relationship quotient is all about commitment, flexibility and top-to-bottom approach path.