
Why .NET Technology is Important for ERP

*The Benefits of ERP Systems Built with
Managed Code*

A WHITE PAPER

By Will Hansen, CPIM
Vice President of Technology
Intuitive Manufacturing Systems



Introduction

The release of Microsoft's landmark technology called .NET ("dot net") posed an unprecedented challenge for all ERP companies. While the public has only recently become aware of Microsoft .NET, among ERP software vendors the key strategic decisions about how to apply .NET occurred back in 2000. The direction each vendor has chosen to go in leveraging .NET technology has cast their fate and stands to impact each of their customers on a strategic scale over the next few years. The manufacturing industry is moving rapidly to a new level of ubiquitous connectivity and inter-system integration. Despite many words to the contrary, some ERP companies and by association, their customers, are at risk of being left behind.

Bad Timing for Many

In the late 1990's many ERP companies caught the web browser wave, undertaking projects to leverage the Internet and browser technology and even to convert their software to "lite-client" or web "portal" architectures. Unfortunately for some, .NET came on the scene too soon after this major overhaul. When .NET appeared, some were too technically exhausted, or inflexible, or still basking in the glow of their new "Internet-based architectures" to recognize and embrace .NET. Other ERP software companies were and continue to be simply too busy struggling to stay solvent during the devastating one-two-three punch of Y2K, the recession of 2000, and 9/11. They lack the resources to consider the complete restructuring of their products that .NET warrants.

While Microsoft struggles to explain and market .NET, the changes at its core represent the biggest shift in software technology since the dawn of Microsoft Windows. .NET is the second shoe to fall in the Internet revolution, portending a whole new computing model emphasizing not just superficial trading of web pages, but cooperating and collaborating systems. ***For ERP vendors and their customers, .NET spells the future of enterprise software applications.*** And, as we will explore in this whitepaper, .NET technology points to a future that demands ERP software companies rethink and rewrite their base architectures.

The Right Place, the Right Time: Accepting the Challenge

In 1998, Intuitive Manufacturing Systems was a 4-year-old ERP company, following the industry leaders, investigating "lite-client" architectures and tools to move their ERP package into the Internet age. Intuitive's investigation turned up several concerns.

First, web browser-based user interfaces would not provide the rich features and raw speed of native code. Native code (code running locally on the PC) leverages the local CPU to perform much of the computing work, giving the user interface a snappy performance, and a rich set of reactions to a user's mouse clicks and keystrokes. In the lite-client model, most of the work is performed on the server. A web browser interface is analogous to a slide show, where the projector (the server) does the work and the screen

(the browser) just hangs there. Clearly, the lite-client processing model mirrored that of the mainframe with its dumb terminals rather than leveraging the nimble and ever more powerful PC.

Also, the “lite-client” architectural model put more strain on the limited IT resources and budgets available in smaller manufacturing plants. It meant ERP customers must use larger server hardware and web server software and maintain IT budgets to keep it all running. They would not be giving up their PCs; web servers would be an additional element to manage.

Finally, and possibly the biggest concern was that moving to a lite-client architecture did nothing to facilitate a rich level of connectivity and distributed processing, the need for which was clearly visible on the horizon. Hardware, networking, and communications companies continue to bring forth smarter and more powerful PCs, hand-held wireless devices, cell phones, and Dick Tracy wrist watches. Lite-client architectures do little to capitalize on these advancements.

When .NET was made public in June 2000, Intuitive was on the spot, literally. Located 10 minutes from Microsoft’s main campus and with a corporate history of using only Microsoft technology, .NET was simply a natural and logical fit. After learning the technology in Microsoft’s .NET Early Adopter labs, Intuitive began to not just use .NET, but to design a completely new ERP architecture to fully leverage .NET. For the reasons we are about to explore, Intuitive realized that .NET warrants a complete and total conversion of every line of code.

Why .NET Software is Better Software

.NET is not another version of Microsoft Windows; it is the next-generation computing platform from Microsoft. At the core of the .NET hype is a new layer of software that sits above the Windows operating system. This new .NET platform is a better platform for building and running applications software than the Windows OS that we know today. .NET insulates software developers and computer users from the deficiencies and incompatibilities of operating systems. And .NET provides a new set of tools and prefabricated components of unprecedented power that can be used to write a new type of software called “managed code.” Over time, managed code will become recognized as clearly superior because:

Managed code is more robust. The .NET platform not only runs this new type of code but oversees its execution allowing software errors to be caught and halted before serious problems can occur. “Memory leaks,” “memory corruption,” and “blue screen of death” problems that freeze our old Windows systems in their tracks are prevented by .NET or curtailed before damage is done.

Side-by-side versions. .NET software minds its own business. It stays together in one place rather than smearing itself all around our systems. And in doing so, it sidesteps the old Component Object Model (COM) -based mechanisms that allowed software packages to collide with and harm one another. In fact, .NET allows two versions of a managed code application to run side by side on the same computer without any interaction or conflict. .NET thus eliminates the need to remove an old version of a software product just to install and try out a new version.

Better security. We all know our systems are susceptible to hackers. Under the old Windows approach to security, software is granted permission to do things on your computer based on the permissions granted to the user who runs the software. Hackers continue to find devious ways to sneak destructive software onto our systems and trick our computers into running that software under a privileged user account. In contrast, managed code follows a security model that is inherently different; the software is granted permissions on its own merits – not those of the user. Attributes of the software itself such as who wrote the software, where it came from, and where it is located are used to dictate and police what the software is allowed to do. The .NET platform enforces the new regime. Managed code can only run on the .NET platform so must submit to this new system of strict supervision and restriction. Certainly hackers will continue their agenda, but the new .NET security model is a promising new weapon in halting rogue software and security breaches. You've heard the old adage that you're either part of the solution or part of the problem. Managed code is part of the solution. Every purchase of old COM-based software is a vote for perpetuating the security problems that haunt the computer industry today.

Better connectivity. The new .NET software development tools and the standards-based nature of managed code make it easier to develop systems that employ state-of-the-art connectivity techniques such as XML Web Services. Our future will clearly be filled with a kaleidoscope of constantly collaborating systems, large and small, networked together, exchanging information and performing services for each other and for mankind. Building this high level of connectivity and requisite robustness demands new components, tools, and a standards-oriented approach. Getting there with old components and tools and without standards is an impossible challenge. .NET comes with a blueprint and toolset for the next generation of software for the new "connected world."

Faster software, faster development, easier deployment. .NET is a completely new software technology, created from scratch, both to leverage groundbreaking technologies such as XML and to eliminate the old inefficient internal layers that have built up over the last 20 years in Windows software. Managed code is lean, speedy, and lightweight. These traits enable new innovation in application development and deployment. And, .NET includes a huge box of prefabricated industrial-strength components ready to use by software developers, allowing developers to save time while creating more robust and

powerful applications. Aspects of managed code coupled with industry standards invite new deployment models including “no-touch” deployment and software that updates itself.

Lower cost of ownership. IT departments struggle daily to fix problems and work around restrictions rooted in the old Windows COM-based software technology. Companies expend vast amounts of corporate energy in their efforts to prevent new debilitating problems as they endeavor to apply system upgrades. Innovation and progress in leveraging new technology is slowed by severe caution and pessimism in rolling out new changes. The problems inherent in old Windows software translate to higher IT costs and lost opportunity costs. Each of the managed code advantages listed above promise to contribute to lower IT costs due to easier software development, easier and more trouble-free deployment, installation, maintenance, and security.

The advantages of managed code are clear and recognition of this fact is gradually growing in the public consciousness and corporate boardrooms. Just as there came a day when the MS-DOS-based software market was pronounced dead, so will come the day when software purchasers will only settle for managed code.

.NET is a new and better kind of software and a new set of tools to build it. And yes, .NET makes it easier to build and deploy the XML Web Services that everyone is talking about. Not to diminish their importance, but the fact that Web Services are dominating discussion right now is a symptom of a problem many ERP companies face.

The Avocado Analogy

From the standpoint of development effort, ERP software packages are structured like a ripe avocado; a soft wrapper around a hard central nut. The soft flesh of the avocado is like the interfaces that surround an ERP system: graphical, user-interactive interfaces including native code and web servers dishing out web pages, and non-user interactive interfaces for connecting to peer systems through Electronic Data Interchange (EDI) and other Enterprise Application Integration (EAI) mechanisms. Want to connect to another system, or have a new visual look? Just modify the interfaces – they’re pretty malleable. In contrast, the hard central core of an ERP system is the software that implements the business logic; the vast collection of rules that define the composition and flow of each business transaction and the rules and conventions for ensuring data accuracy, integrity and appropriateness. Business rules protect the corporate database stronghold, preventing illogical data from the outside world from corrupting the company’s information and ruining its value. Business rules are the heart of an ERP system – more valuable than the database design since they are more difficult to create and maintain. They, and the framework in which they are implemented, are not casually altered.

To get a better handle on business rules as the crown jewels of an ERP system, consider the example of a seemingly simple inventory withdrawal transaction. Let’s say a quantity

of a particular inventory item gets pulled from stock. What happens? Certainly, the ERP system decrements the inventory on-hand balance. Another business rule dictates that a record of the transaction be written to an inventory transactions history table. Still other business rules come into play to perform credit and debit postings to the general ledger. Probably a record will be written to reduce or extinguish the expectation to withdraw inventory if the withdrawal was a planned event which has now taken place. Additional business rules handle any lot tracking or serial number ramifications. If the withdrawal occurred as a result of shipping the item against a customer order, then many other business rules come into play. Those rules would write a shipment log, reduce the “remaining to ship” quantity on the sales order, trigger the invoicing system and accounts receivable, send a message to alert the customer, update the sales history of shipments to this customer, calculate cost of goods sold, write necessary records for salesman’s commissions, etc. Which GL accounts are to be posted? How are costs to be calculated? What salesmen get credit? Any taxes get paid? What are the lot numbers involved? Business rules cover all of these issues and more, so that when, in this case, an inventory withdrawal happens, all the appropriate subsystems and tables that make up an ERP system are properly notified and data accuracy and completeness is maintained.

Such rules are the core of a modern ERP software system – they are the source of the value that an ERP system is expected to deliver. Most of the development work and testing of an ERP system is spent making sure that these rules are correct, complete, properly invoked, and that all transactions work smoothly and efficiently. All but the most superficial software features in an ERP system are implemented as business rules. If the database is the heart of an ERP system, the business rules are the soul.

Something to Hide

In many ERP packages today, the real work is still done by a body of old and dated software; business rules written long ago in old software languages. These old jewels have been coddled and insulated from decades of technology change by sandwiching them between layers of newer software. Old software born on obsolete operating systems can be literally tricked into running on a modern Windows OS by inserting a layer of new code beneath it. And it can be concealed from the outside world by wrapping it in layers of modern graphical user interface and EAI code.

So, software written during the 1970’s and 1980’s resides at the heart of many corporate systems. Its designers certainly didn’t know about Microsoft .NET, XML, 2.3 GHz processors, and all the other technologies and standards we have today. They knew about procedural languages, 8 MB of memory, 1200 Kbps modems, dumb terminals, and raised floors. But the investment in business rules is huge, then and now, and that old code still seems to do the job. So, when it comes to the central cores of ERP systems, “if it ain’t broke, don’t fix it” has been the law of the land. Or should we say: “don’t update it, just wrap it.”

Ironically, the move to using Internet browsers for the user interface was a god-send to some companies with aging business logic code. Saying you had a web browser interface sounded pretty progressive from a marketing standpoint. Confessing that your old code was wrapped in a “screen scraper” did not. But technically, the result was the same; a core of old software given a more modern face.

And later, when .NET appeared it was relatively easy for ERP companies to replace their wrapper software with a new .NET equivalent and fly the .NET flag. The same goes for XML Web Services. In truth, behind the web pages and beneath the new masks built of managed code still reside large amounts of old pre-.NET, and often pre-1990's software that continues to plug along. These cores of old software are skeletons in the closet and the benefits of managed code stand to spring the closet doors wide open in coming years.

Now, let's set the avocado analogy aside and look at another dimension of the problem of old software architectures.

Two is Not Better than One

The core architectures of most ERP systems were designed and written long ago, before the demands of the new connected world were understood. Connectivity and EAI support have literally been afterthoughts, considerations for which have been applied carefully at the margins of the old core software.

By definition, business rules must be comprehensive and accurate. The last thing one wants is to have to maintain multiple sets of them, but that is what some ERP companies do. The foundation of an ERP product is the main set of business rules that handle each type of transaction when it comes in from the user working at a keyboard. A separate implementation of those rules may exist to handle the same transaction when it arrives via EDI or over the web or from a hand-held wireless device on the shop floor.

It's typical in EAI strategies today to have the integration mechanism, Microsoft BizTalk for example, perform its own data validation and apply some or all of the business rules before passing the data to the destination system. Therefore, business rules that exist within the destination system are essentially duplicated within the EAI mechanism. This doesn't happen for free. Such duplication of logic increases the cost of integration, slows the evolution of the software system, and makes connections to other systems more fragile and costly to maintain.

The cost of duplication of business rules is hidden in IT and EAI budgets and ERP software price tags. Maintenance and enhancement costs are inflated by this duplication, but more importantly a company's flexibility and agility is severely restricted. Any new change to a particular business rule must be made in all the places where that rule exists.

It usually takes time simply to find all the places where a given rule comes into play. Keeping them all synchronized as business evolves is a difficult task.

We live in a fast-changing corporate landscape. Mergers and acquisitions and corporate divestitures constantly scramble business relationships and corporate standards. Technology, industry standards, and best practices are also constantly evolving. The software that implements the business rules must change as the business changes. And the pace of change is increasing, not slowing. ERP architectures of the past fight against change and are a major hindrance to agility. They stand in the way of the new world of nimble integration and ubiquitous connectivity.

ERP Architecture for the 21st Century

Today, too much focus is on “getting connected” and on the communication from the wrapper or outside surface of one system to the outside surface of another. Transport and communication technologies such as web services, message queues, EDI, and BizTalk get all the attention. The issues of old core code and duplicate business logic are not spoken of; it’s a secret problem that most systems have. It’s assumed that n-tier architectures with their emphasis on shared middle tier business rules are used and that’s all that needs to be said. But many aging n-tier architectures are connection-challenged and require duplication of business logic to get by. Looking to the future, the entire ERP architecture should be designed with EAI considerations in mind. An ERP system for the 21st century should be “born to connect” rather than “coerced to connect.”

Modern software architecture for ERP is all about efficiently implementing business rules. It’s all about how those rules are packaged and invoked within the system and how easily they are exposed to partner systems. That’s why .NET and the new standards-based world warrant a redesign of ERP systems from the ground up. A checklist of the characteristics of a modern ERP system must include:

100% managed code. Truly modern software packages should be completely constructed of managed code. Only then can they enjoy side by side capabilities, new security, and all the other benefits and lower IT costs that .NET makes possible.

Single set of business rules. Systems should package business rules in such a way that duplication is avoided. Ideally one copy of each rule should exist and all transactions from all sources use that one instance. One set of business rule objects should be able to handle all comers; transactions arriving from trading partners as well as the ERP system’s own user interface, whether it’s a powerful PC, an internet client, or a wireless hand-held device. The rules must properly recognize when operating in a user-interactive situation to support a rich user interface experience and also operate in isolation when handling batches of data arriving with no user around to display error messages to. The clear advantage of a single set of rules is much lower cost of

development and maintenance and greater speed and agility in forging connections to other systems.

Bundle validation rules with processing rules. A key to building a powerful single set of business rules is the ability to keep the logic that tests incoming data for validity readily accessible to the rules that perform the processing and database updates for each transaction. Allow the very same validation logic that serves the user interface to be invoked to protect the system in non-user-active situations such as EDI and BizTalk scenarios.

Dynamic choreography of business rules. A desirable trait in an architecture is that it allows business rules to act like a network of specialists, each handing off at the limits of its expertise to another specialist to take over. Each specialist knows nothing of the actual work performed by its associates. The architecture carefully orchestrates the transaction processing to bring each specialist into play, to do its special work, and hand off to the next. A powerful architecture lets all of this autonomy happen, yet the whole transaction can gracefully unwind and roll back if a problem arises.

Support rich and lite-client interfaces. Neither rich client nor lite-client interfaces are superior – each has its place. The architecture should provide strong support for both native code and web browser-based user interfaces without any development time penalty, thus allowing the software developer the freedom to select the appropriate tool for the job.

Distributed processing. The architecture should allow processing to be completely accomplished on a single CPU or distributed across many systems to spread the processing load. In other words, provide both “scale-up” and “scale-out” capability.

Readily customizable. A modern architecture should provide a framework in which business rules and user interfaces can be easily tailored to the individual company and where those customizations are kept isolated from the standard business rules and UI code, allowing software upgrades and easy maintenance of customizations. When a single, universal set of rules exists, changes can be implemented by revising the one single instance of the company’s custom rules.

Native support for standards-based connectivity. XML is now the indisputable standard for data exchange. An architecture for the 21st century should speak XML when conversing with the outside world, by publishing its inbound data formats in XSD and receiving inbound data as XML, and by sending outbound data in XML accompanied by schema. Internally, data flowing through the system that is visible as XML opens the door to innovation. However, the system should also be able to easily accommodate bi-directional exchange of delimited text files with older systems.

Loosely coupled. Interactions with outside systems should be at the level of granularity of the business document: invoices, sales orders, forecasts, shipping notices, etc. Some ERP systems today expose only a fine-grained interface to the outside world that requires intimate knowledge of the ERP system on the part of the outside party. Such knowledge, while allowing a close connection, causes the interface as a whole to be fragile and more costly to create.

Expose a rich standards-based set of data to the outside world. While separate business entities should exchange data at the more formal level of a business document, the architecture should also allow outside parties to answer their own questions by obtaining detailed data through an open, standards based mechanism. In short, under security control, any ERP data or process should be readily exposable through XML Web Services. The architecture should expose familiar business logic objects for major entities and transactions in the system: Customer, Customer Statement, Vendor, Vendor Performance, Item, Inventory, Work Order, Purchase Order, Project, etc., allowing trading partners to readily find and retrieve the information they need.

Single focal point for inbound data. For inbound documents, the architecture should expose a single “catchers mitt” supporting EDI-style batch transactions and “batch of one” transactions. Such a single entry point should accept all inbound XML documents that are in the proper (published) format arriving from a valid and permitted source. Integrators simply pass an XML document of the proper format (XSD Schema) or equivalent text file and let the business logic be invoked to either successfully processes it into the system or suspend it, awaiting error review and correction. Documents can come from BizTalk or any other Enterprise Application Integration (EAI) mechanism.

Provide event-based communication. Besides supporting deliberate bi-directional data exchange with trading partners, the architecture should fire events which can be listened for by trading partners. When situations arise or certain conditions are met, collaborating systems can respond to each other's needs. In other words, the architecture should let trading partners ask for or pull what they need, and also push transactions and event-based messages to them when situations warrant.

Summary

Microsoft .NET is tangible evidence that commerce in the 21st century will be all about rich and often ad-hoc connectivity, leveraged by collaborating systems running on the ever smaller and more powerful hardware devices we've already come to expect.

Microsoft .NET is a wake up call, and some have missed it. To flourish in the coming years, mission-critical corporate software systems such as ERP must be designed from the ground up for connectivity and integration, delivering maximum corporate agility at the lowest possible cost. An enabler of such agility is a software architecture that represents

business logic in a way that prevents the need for multiple sets of business rules to support multiple sources of transactions.

.NET is also about better software. Managed code is faster, smaller and lighter, it's more hacker-resistant, and easier to deploy, install, maintain, upgrade, and if necessary, remove from your computer. Managed code side-steps the root causes of many of the bugs and security problems we suffer from today. Managed code will be cheaper to own and therefore of strategic value in the years ahead.

Beware pretenders. Commercial software vendors don't advertise their shortcomings. ERP systems are not reinvented quickly and systems today are not well architected for the next phase of computing and communication. Many techniques, including Web Services allow old code bases to perpetuate their masquerade as modern software systems. Discriminating purchasers of ERP software must shop carefully to purchase systems that fully exploit technology to deliver the highest agility and the lowest cost of ownership.

About the author:

Will Hansen is the vice president of technology for Intuitive Manufacturing Systems. He has more than 20 years experience in developing commercial MRP and ERP software packages, holds a Bachelor of Science degree from the University of Washington and CPIM certification from APICS.

About Intuitive Manufacturing Systems:

Intuitive Manufacturing Systems' intuitive, flexible enterprise solutions help small to midsize manufacturers improve operational efficiency and profitability by integrating business processes on an enterprise-wide level. Since 1994, Intuitive has been a technology leader, dedicated to providing manufacturers software solutions that add value to their business. Intuitive's enterprise solution is designed to deliver superior functionality, rapid implementation, high levels of usability and a technology standard unmatched in the industry. For more information about Intuitive or Intuitive ERP, please visit www.intuitivemfg.com.

Copyright © 2003 Intuitive Manufacturing Systems, Inc. All rights reserved. Intuitive ERP is a trademark of Intuitive Manufacturing Systems. Microsoft is a registered trademark of Microsoft Corporation in the U.S. and other countries. All other trademarks are the property of their respective owners.