

Enterprise Application Integration

Shaping the Reality Today

By: Michael Crouch

Changing Landscapes

The current marketplace is an ever growing and changing landscape. New applications, new technologies, and corporate acquisitions and mergers provide more challenges everyday. Added to that set of challenges is the need to leverage legacy systems and data, as well as the need to make more data available to the end-user or customer, and the challenges are monumental. These challenges have created a growing trend in the market for a more formal methodology to make these dreams a reality. Enterprise Application Integration (EAI) is the process by which corporations are able to tie together more than one system or data source. It addresses a problem developers have been trying to solve ever since we began moving applications off centralized processors.

Chances are, if a corporation has been around for a while, they have generations of systems using all sorts of enabling technology that have been built or bought over the years. These may include mainframes, UNIX servers, and Windows NT servers. With the increasing popularity of packaged applications—such as SAP, PeopleSoft, and Baan—it is easy to see the intensified need to integrate those existing systems with other custom and packaged systems.

EAI is really a backlash to traditional distributed computing. Now that generations of developers have built these automation islands of information and business processing, more users and business managers are demanding seamless bridges between them. While the business case is clear and easy to define, the task of implementing a solution is not. There is a change in the landscape from building stand-alone applications to building integrated applications that can take advantage of data and business processes from legacy systems. This paper will try to provide the background necessary to understand Enterprise Application Integration and the challenges customers face in moving towards an integrated solution.

Software Architecture

All systems deployed have some form of architecture, some better than others. Until recently, software architecture and application design were heavily intertwined. The software architecture was implicitly defined into the application design. The reason for this intertwining was a focus on modularity and a structure that would provide basic building blocks for the application. This approach has proven to be an inward-looking approach rather than an outward-looking approach that focuses on the application *and* the system in which the application will run. It is this enterprise wide system view of the application that has been missing from many of the application development efforts in years past.

Building applications that can work collaboratively with other applications within an enterprise is something that must be planned for early in application development. That planning may be something such as providing a robust Application Programming Interface (API) that will allow other applications to access key data or business processes through a C interface. It may be more complex, such as building the application based upon some distributed computing model such as Microsoft's Component Object Model (COM) or the Object Management Group's CORBA. These models provide a basis for developers to build applications that are both modular and have a well-defined structure.

However, while modularity and structure of components is necessary, it is not sufficient for achieving flexibility in integration. Even with modularity, it is possible to have a tightly coupled system where the components become too interdependent on one another. The flexibility in integration provided by the modularity and structure of the application can often be lost through the process of change when interfaces and component implementation are too tightly coupled. Separating the software architecture from the application design is the key to achieving the required flexibility. This is even more evident when application integration is considered. At this point the architecture of the solution becomes essential.

History of Integration

Most of the initial attempts at Enterprise Application Integration have focused primarily on problems with data synchronization. The primary function of the EAI system in these cases is to ensure that some data used in a given application can be replicated, and propagated fairly quickly to other applications. The driving need in most of these systems is usually a process that requires manual intervention or some measurable time to complete. These processes need to be completed in seconds or minutes, instead of hours or days. The need for operators re-key the same data into multiple systems is time consuming and error prone. While technologies such as asynchronous messaging help create these new business processes, the remaining components of the system are employed primarily to benefit the IT department.

Providing a uniform mechanism to control processes and a set of tools to define the data sources, transformations, and routing simplifies the overall development and maintenance of these relationships. Also, by centralizing what had historically been a distributed task, a common method can be defined and implemented. This solution leads to a reduction in the overall cost of development and maintenance of the integration. Additionally, some level of automation is added, which may result in a reduction of errors and increase in productivity of the system.

The second form of Enterprise Application Integration, which has been attempted by only a few companies, involves implementing some form of process integration. This type of implementation focuses on the automation of business processes. It is more complex since it requires additional capabilities beyond basic messaging, transformation, and routing. This type of integration also includes process management, business-process integrity, and complex process flows. Early efforts in this area have yielded significant benefits beyond the IT organization. These integrations have extended the level of integrated process automation, which many business users have come to expect, into domains where they aren't currently available. In effect, these integrations have helped to push the processes beyond the traditional user to the casual user and in some cases to the consumer.

These integrations also blend legacy systems into the overall automation without any additional investment to rework those environments. The result can be a highly automated and flexible process, which can ultimately become a key business differentiator. As an example, integration between Order Management (via the Web) and an existing Order Fulfillment application is often the initial project of this type. It requires many different capabilities in the integration system, some of which are quite complex and challenging to implement. The result can be significant for the corporation that is able to produce such an integrated system.

The third, and most complex, form of Enterprise Application Integration is designed to connect internal processes to processes that exist in systems outside the enterprise. Even though this type of integration has been available in some rudimentary forms for some time, the emphasis has largely been on sharing data between companies, and not processes. This sharing of data continues to be one of the most common areas of integration beyond the walls of the enterprise and can be seen in the increasing interest in the Extensible Markup Language (XML). Most practical forms of this type of integration are based on electronic data interchange, and are not truly integrating business processes yet. One of the key factors preventing process integration is that process automation assumes significant time lags between organizations, and so integrations need to be built that can take this factor into account.

The industry has yet to get the plumbing right to really solve the multitude of problems that EAI is intended to address. While many vendors claim to possess the ultimate EAI solution, there's a long way to go before developers can effectively integrate all applications within an enterprise with a single tool. There are still some lingering problems that include things like application interfaces that require new programming and maintenance. There is no single solution that binds applications at both the data and method levels for all of the different technologies used in deployed applications. So, as it stands today, customers will need to use packaged EAI solutions as well as do some application programming. Whether that is wrapping an existing application, modifying an application to provide a usable API, or building new adapters to applications or technologies, there will always be the need to do some development. Many organizations consider that a high-risk approach, and are looking to minimize that risk wherever possible.

Still a Long Way to Go

Enabling technologies such as XML will help to simplify some aspects of inter-company integration. Still some issues such as definition and administration of common processes, rules governing the ownership and stewardship of data, and processes for error detection and recovery all remain extremely difficult. There are very few solutions that address these remaining challenges. However, the benefits of this type of integration, such as reduced inventory and cost in the supply chain and significantly improved service levels, are too compelling to ignore.

Enterprise Application Integration will have a significant effect on technology, architecture, methodology, and structure of the application delivery organization. From an architectural perspective, EAI introduces a new architectural component to the mix — the integration server, which provides the services needed to accomplish the integration. As solutions evolve, multi-hub and distributed systems will create topologies of increasing complexity, which will be handled by the integration server.

New application design will also be affected, as it's no longer sufficient to assume that applications can communicate via Structured Query Language. Higher-level interfaces, exposed through component interface models such as Microsoft's Component Object Model (COM), the Object Management Group's CORBA, and Sun's Enterprise Java Beans, must be created.

Companies that wish to successfully implement some form of application integration will need to develop new skills required in the EAI arena. These skills include things like messaging, asynchronous communication, and new ways to maintain process and data integrity while working with the new asynchronous relationship between applications. They will work to define new mechanisms, including process integrity and flow management and compensating actions. Their development staff will be conversant in these new concepts and new technologies, as well as new tools for defining processes and process integrity, data transformations, and application interfaces.

An Evolutionary Approach

The implementation of changes needed within an organization to take on an EAI solution can, and should, be coordinated with the phases of the evolution of Enterprise Application Integration itself. In the early stages, the requirements should focus on developing application interfaces and creating reliable information transport mechanisms. Simply integration of data will help most organization realize some initial benefits. As more complex process automation is desired, the application interfaces and transport mechanisms can be evolved to meet those needs. Once these issues have been addressed within the enterprise, the focus can move to addressing integrations with external organizations. At this point the companies must have clearly defined processes, infrastructure, and organization in place to manage the rapid and possibly conflicting demands of a large group of business partners.

The transition through these phases will likely result in some type of transformation within the company. Typically you can expect to see the IT organization change from a technology-centered to a more of a process-centered model. The formation of process teams will take place, and they will become one of the core units within IT for managing distributed automation. The formation of an architecture board may take on the responsibility of defining the integration architecture design and then enforcing the implementation. What is most important in each of these areas is that there needs to be one unified vision as to what the integration is going to provide, what is to be integrated, and then how it is to be implemented and maintained.

Every one of these changes will have subtle but significant impact on the entire organization. The overall benefits of a well-architected approach to integration are varied and significant. The increasing pace of business will demand that IT departments abandon their high-risk, piecemeal approaches in favor of more structured, well thought out approaches. This transition will require changes to the skills, structure, and technologies employed by the IT organization. Companies will look for ways to help ensure this transition takes place correctly and puts them on the correct path to achieve their integration needs.

Levels of Integration

The opportunity for integration can be found at a number of different levels within the enterprise. At a very high level we can say that there are distinct levels of Enterprise Application Integration; Data Level, API Level, Method Level, and Application Level. Each of these different levels of integration has their advantages and reasons as to why they provide the best solution to an integration problem. The following paragraphs provide some deeper insight as to what the integration involves and why it would be selected.

Data Level EAI

Data-level EAI involves moving data between two or more databases in order to integrate the applications. Data level EAI is used to extract information out of one or many databases, perhaps transforming the data (either format or content) so that it makes sense to the database receiving it, and then placing the data in the target databases. The concept is to avoid changing the application logic while moving data between applications, thus eliminating the need to re-write, test, and deploy the applications. The fact that the application logic remains untouched makes this method of application integration very attractive. The difficulty is that the access to the business processing logic is still through the application, and so the middleware must be able to handle that processing logic.

Data-level integration provides simplicity and speed-to-market, and is typically cheaper to implement than other forms of EAI. Most applications and interfaces typically were not designed to accomplish EAI, but rather to work independently. As a result, to successfully implement data-level integration requires bypassing the application's logic and extracting or loading the data directly into the database through its native interface. Fortunately, most applications decouple the database from the application and interface, making this a relatively simple task. However, this doesn't mean that data-level integration is always a good idea. First, you must consider how tightly coupled the data is to the application logic. Moving data between databases without understanding the entire application is always a dangerous maneuver.

Given the conceptual simplicity and real life complexities, how does an enterprise implement data-level EAI? Ultimately, it comes down to understanding where the data exists, gathering information about the data, and applying business principles to determine which data flows where, and why. Many are compelled to create an enterprise-wide data model complete with meta-data (data about data) to help EAI architects and developers better determine source data, as well as the potential target systems.

API Level EAI

Like data level EAI, API level integration avoids the cost and risk of having to change the source and target applications. This is accomplished by using application program interfaces (APIs) that exist within the applications. Using these interfaces, developers can cohesively bind applications together, letting them share business information and business logic in a noninvasive manner. The only limitations facing developers are the API's specific features and functions, which vary from pretty good to nearly useless. SAP R/3's Business Application Programming Interface (BAPIs) is an example of an API that can be used to integrate other applications to SAP.

Simply put, APIs are interfaces that developers expose from an application to access the various services or data elements. Some interfaces are limited in scope, while others are feature-rich. As an example, Microsoft Word exposes a COM interface that allows other applications to integrate to Word. Through this interface, developers have access to a number of different methods that implement most functions available on the Word application menus. Typically these types of interfaces provide access business processes or the data directly, and sometimes both.

Application interface integration as a means of EAI is distinct from method level and user interface level integration in a number of ways. In method level integrations, it is possible to distribute the exposed methods that exist within enterprises among various applications. These methods are typically shared via common business logic-processing mechanisms, such as application servers or distributed objects. User interface level integration makes business processes and data available through the user interface instead of through an API.

These differences distinguish application interface level integration from other types of EAI. The potential complexities of the application interfaces, as well as the dynamic nature of the interfaces themselves, simply add to the difference. Because interfaces vary widely in the number and quality of the features and functions they provide, it is nearly impossible to know what to expect when invoking an application interface.

Packaged applications, which are most often present in a typical EAI problem domain, are just beginning to open their interfaces to allow for outside access and, consequently, integration. While each of these packaged applications determines exactly which interfaces should be available and what services they will provide, there is a growing consensus in providing access at the business model, data, and object levels. When accessing the business model, or business processes, developers typically invoke a set of services through these interfaces. For example, access to credit information can be gained through the user interface by driving the screens, menus, or windows. Invoking a method through the API provided by application may also access this information.

When you take a look at custom applications, anything is possible. If there is access to the source code, it is possible to define a particular interface. For example, rather than accessing the user interface (screen scraping) to reach an existing COBOL application residing on a mainframe, you can build an API for that application and expose or extend its services. In most cases, this requires mapping the business processes, once accessible only through screens and menus, directly to the interface. The downside to this approach is cost. In many cases, it is more cost effective to simply access the application information by automating access to user interfaces from a program through a user interface level integration.

Method Level EAI

Method level EAI integrates applications by binding them together at the method level. This is by far the most invasive form of EAI, requiring that major changes to the application source code. This can be done through any number of traditional application method-sharing technologies such as distributed objects or application servers. It generally means creating a hybrid or composite application, which can provide the infrastructure for accessing shared business processes.

Attempts to share common processes have a long history, going back to the early days of distributed objects and multi-tiered client servers. The idea is that a set of shared services on a common server that originally provided the infrastructure for reuse, now facilitate integration. Reuse is important in this context. By defining a common set of methods, you can reuse those methods among enterprise applications. This significantly reduces your need for redundant methods and applications and thus opens up these methods as business processes that can be shared across applications.

Using the EAI tools and techniques to integrate an enterprise not only helps you share common methods, but also provides the infrastructure to make such sharing a reality. By integrating applications so that information can be shared and providing the infrastructure for reuse, we achieve both reuse and application integration. This might sound like the perfect EAI solution, but the downside is that it's usually the most invasive level of EAI. Unlike Data Level and Application Interface Level integration, which don't typically require changes to the source or target applications, Method Level EAI requires changes to many, if not all, enterprise applications.

The enabling technology for method level integration is a product (or standard) that allows applications invoke other application's methods or access shared methods on a central server. Examples of this technology include distributed objects, based on CORBA or COM+, as well as the new breed of application servers. Sharing common business logic between many applications is a tremendous opportunity. However, it comes with the risk that the expense of implementing it will outpace its value.

User Interface-Level EAI

User interface level integration leverages the application's user interface as a noninvasive point of integration. As an example, if there was a need to integrate existing mainframe applications with other applications, and there are no other points of integration, such as the database or API, it may be possible to access the data through the application's 3270 terminal interface. This process typically uses windows and menus to access the relevant data that must be extracted and moved to another application or data store. While it sounds like an inefficient and perhaps even ill conceived approach, there is a great deal of it going on in application integration today.

Some of the other EAI levels may be more technologically appealing and efficient, but in many applications, the user interface may be the only available mechanism for accessing logic and data. In spite of its inefficiency in getting into an application, the user interface has the advantage of not requiring any changes to the source or target application. It also lets the integration leverage the business rules in the existing application instead of rewriting the rules at the middleware layer.

It may be argued that User Interface Level EAI should be the last-ditch effort for accessing information from older systems and integrators should turn to it only when there is no well-defined application interface. But integrators should not avoid User Interface Level EAI completely. In most cases, it successfully extracts information from existing applications and invokes application logic. Moreover, if implemented correctly, there's virtually no difference between the user interface and a true application interface, except maybe in the efficiency.

The enabling technology for user-level EAI has been around for years. Most 3270 emulators, for instance, provide APIs to access screen information. With all of the attention devoted to XML as the new standard for sharing data between corporations, it is clear that the value of the data presented at the user interface is being recognized by industry.

Stages of Integration

Application integration can be any number of things to different customers. Integration can be broken down to a few key stages that typically coincide with the evolution of application integration at most corporations. Each of these stages provides real value to the company and provides a clear path to becoming more successful and competitive within their markets.

The first stage is data synchronization. In this stage the primary focus is to ensure validity of data as it is propagated between systems. As corporations are more capable of facilitating unidirectional data flow, they become more efficient in their handling of data. This helps reduce what is called the redundancy factor where data must be manually entered into a number of different systems. In this type of integration, the data flows between applications so that similar data in separate application is consistent.

The second stage is process synchronization. The focus of this stage is to provide a level of automation in the end-to-end processes. The challenge is to manage process integrity and provide multidirectional information interchange between applications.

The third stage is external integration. In this stage the goal is to furnish data and process flow outside the enterprise. That requires corporations to standardize their shared external processes and demands distributed administration to make sure the processes are available and working correctly.

The Integration Process

In David Linthicum's book, "Enterprise Application Integration"¹, the proposed process for application integration is presented in terms that are easily identified and understood. The process draws upon familiar concepts that include analysis, modeling, requirements gathering, meta-data, and schemas. I would like to presents a modified version of his twelve-step approach to application integration that is a bit more basic and somewhat easier to understand.

1. [Understand the Problem Domain](#)
2. [Make Sense of the Data AND Processes](#)
3. [Identify Key Integration Points to Applications](#)
4. [Identify Business Events](#)
5. [Identify Transformation Scenarios](#)
6. [Map Information Movement](#)
7. [Apply the Right Technology](#)
8. [Test, Test, and Then Test Again!](#)
9. [Optimize and Tune for Performance](#)
10. [Maintenance Procedures](#)

Understand the Problem Domain

In order to solve a customer's problems, you must first understand their business. It is crucial to ask lots of questions to get a clear understanding of their problems in the context of their business. Integration problems do not exist within a vacuum, but rather within the confines of the physical system and the processes that drive the business. These processes are what help define the enterprise in the reality of their organization. You must first understand *how* they do business to be able to determine what is and what is not important.

Make Sense of the Data AND Processes

It is important to understand that it is the data that drives application integration. Without data, there is little need to integrate. To understand the data, you need to know where it exists, the format and structure (i.e. schema definition), data flow definitions (what flows to where), and why the data flows exist. Many times it is useful to build an enterprise metamodel to document the enterprise data for use in subsequent steps.

The processes are a key component in this step in that the logical connection between the data and the business can be understood. A process model may help the integration team better what the business process do, and therefore make it easier to identify process integration points. It will also shed light on how processes interact, with data and with each other, so that the integrity of the existing application is not compromised in the integration solution.

¹ David S. Linthicum, *Enterprise Application Integration*, Reading, Massachusetts, Addison-Wesley, 2000

Identify Key Integration Points to Applications

All applications are not created equal. That being understood, the focus in this step will be to determine what types of integrations will be possible for all of the applications. As discussed earlier, there are four levels of integration; data level, API level, method level, and user interface level. The integration team must evaluate which of these types of integration strategies are available for each application, and what advantages and disadvantages exist for each technique.

Identify Business Events

For every action (or in this case - event), there is a reaction within the system. In this step it is important to determine how the applications respond to the business events. This is best understood by a technique used in Object Technology known as Use Case Modeling. Use Cases simply provide a mechanism to capture details about external events upon a system, and the behavior of the system in response to the event. The events are typically asynchronous in nature, but can be synchronous in some cases. A complete discussion of Use Case Model is beyond the scope of this paper, but there are many books and articles that provide a detailed discussion.

Identify Transformation Scenarios

After the data and processes are well understood, it is necessary to determine how the schema and content of the data that moves between systems needs to be transformed. This is almost always required to allow the data from one system to be understood and utilized by another system. The fact that the integration is required between two systems, which were developed independently, is reason enough to expect data mismatches (either format or content) to occur. The final word on this is to *expect the unexpected*.

Map Information Movement

After all of the steps above have been completed, the integration team should have a good idea of what the integration effort is. This step is the design phase of the integration. We need to determine, and document, the source information (which data element is moving through which interface in which application) and the destination information (which target data element through which interface in which application). Things to be defined for each case include; physical location of the data, security issues, interfacing methods (on source and target), any business processes involved (either directly or indirectly), and any conditions or events that cause data movement.

Apply the Right Technology

Once the analysis and design phase have been completed, we are ready to enter into the implementation phase. There's a wide range of technologies available to accomplish integration. Most integration solutions will likely require a mix of technologies to implement the solution, which is most often driven by the type of integration levels that have been selected. The key is to find a product (or group of products) that provides an optimal solution for the integration task at hand. No single vendor solves all of the problems, so it is probable that there may need to be a mixture of products. The key is to limit the number and select products that are open and extensible.

Test, Test, And Then Test Again!

Testing is a time consuming and thankless task. Ask yourself when was the last time you called the phone company to thank them for a dial tone. Then ask yourself when was the last time you called to tell them about a problem. Get the picture? It is essential that a comprehensive test plan be developed and implemented in any integration project. If the integration cannot guarantee data integrity, process integrity, system availability, and system performance, then the integration could cause a loss of business or data. To make matters worse, most of these systems are production systems that cannot be taken off-line, so testing becomes a bit more difficult.

Optimize and Tune for Performance

As a word of caution, you cannot code in performance as an after-thought. It must be designed in from the start, and should be tested as early as possible in the implementation phase. The testing phase is far too late to correct a design flaw that is causing performance problems. In some cases the integration may need to be tuned by moving components in the integration to different servers, or optimizing a simple algorithm. The goal here is ensure the integration is running at an optimal level.

Maintenance Procedures

One of the areas that is often overlooked in any software engineering project is the use of maintenance routines. This concept should be addressed in all application integration projects to help in the testing as well as maintenance phase. Since the EAI solution will represent the heart of an enterprise, and the applications themselves are stand-alone applications with stand-alone maintenance procedures (if you are lucky), it is imperative that an integrated maintenance solution be implemented as well. Imagine an application integration solution that started losing vital information as it moved throughout the enterprise, and to isolate the problem you had to interrogate each system using the stand-alone procedures. That might work, if the problem was in the stand-alone application. But what if the problem is in the integration code? As you can see from a very simple example, integrated maintenance procedures are a necessity.

Integration Project Pitfalls

There are many Enterprise Application Integration projects that fail or are scaled back for a number of reasons. Some of the reasons include things beyond the control of the integration team such as the lack of documented, well-understood, and accessible applications. Corporations that are looking at Enterprise Application Integration must ask themselves how to best integrate the applications. There are a number of choices for what level the integration is to take place as discussed earlier.

A combination of integration approaches to solve the many issues surrounding the technical aspects of Enterprise Application Integration is not atypical. There may be good reasons to choose data layer integration for a set of applications, but API layer integration provides a better solution for some other applications. The driving factors in making this decision are what is needed from the application, what integration points are available, and what should the integration achieve.

In the final analysis though, if the applications are not well understood from a process and data standpoint, the odds against success become greater. If you take any SAP installation as an example, it would be foolish to even attempt integrating SAP and any other applications if the SAP installation was not very well understood. The ERP packages tend to be extremely large and very complex in their data storage and business rules. They also vary widely from one installation to the next. Without the detailed knowledge of the way the system is used and what data can be shared with other applications, the chances for a successful integration effort is fairly low and rests more on luck and brute force coding than on engineering. The key is to take on this task in manageable pieces to help insure success.

The Hardest Part

The hardest part of any Enterprise Application Integration would have to be designing an architectural framework for the integrated applications. In short, an architectural framework is a style and method of design and construction or strategic policies and patterns that shape a system — a decoupling of access mechanisms, common services, and facilities from the applications that allow an orderly mechanism to share data and processes between applications.

To better understand the importance of a good architectural framework for integration, consider the following attributes as a way to measure the relative “goodness” of a given architecture:

- Manage change and complexity
- Adapt to system extensions and new application integration
- Minimize interdependencies among complex applications

Architectures come in all shapes and sizes and vary widely from the traditional hub-and-spoke architecture to distributed node architectures. The import point to consider is how well will the proposed architecture meet the needs of the customer and how resilient will that architecture be in the changes it will have to endure over time. Architectures that manage changes and complexities over time provide a clear example of the value they bring to an enterprise.

Conclusion

It is clear that there are some very difficult challenges facing corporations looking to enter into an application integration effort. It is also true that there are a number of competing products within the marketplace that claim to be the one-stop solution to all the integration needs a customer may have. The cold reality of real world experience tells us that if that were the case, there would surely be a dominant leader in this market today, which is not the case. The best solution for one customer may not be the best for another and that decision is driven primarily by the applications they are looking to integrate and what they expect to gain from the integration. With new technologies coming into the markets all the time, it would be difficult to imagine one vendor supporting everything.

The key points to note from this paper are that application integration is difficult, it involves many technologies, requires integration at a number of different levels, and what solution is selected must be open and extensible to changes in the technological landscape. The prediction is that with the challenges faced in the EAI market, vendors will start to provide clean integrations between their solutions and some of their competitors in an attempt to address the wide range of problems customers face. Products that provide open and easily extensible solutions provide the greatest opportunity to solve these problems today, but there is much work still to be done.

About the Author

Michael Crouch has over 18 years of industry experience in designing, developing, supporting, and managing software applications and is the president of MANAGED BUSINESS CONSULTING, Inc., a technology consulting firm specializing in providing business solutions that help clients address their critical application needs. He may be reached at Managed Business Consulting, Inc., P.O. Box 6729, Ellicott City, MD 21042, by phone at 410-750-0923, by e-mail at mcrouch@mbconsulting-inc.com, or on the web at www.mbconsulting-inc.com.